



STIC Search Report

EIC 2100

STIC Database Tracking Number: 169068

**TO: Michael Pham
Location: RND-3D18**

**Art Unit: 2167
Monday, February 06, 2006
Case Serial Number: 10/627,191**

**From: Lance Sealey
Location: EIC 2100
RND-4B11**

Phone: 571-272-8666

Lance.Sealey@uspto.gov

Search Notes

Dear Michael,

Here are the result of your search request.
Please let me know if you have any questions.

Lance

178592



STIC EIC 2100 Search Request Form

Today's Date: _____ What date would you like to use to limit the search?
Priority Date: 4/7/2003 Other: _____

Name <u>Michael Pham</u> <u>81563</u>	Format for Search Results (Circle One): <u>PAPER</u> DISK EMAIL
AU <u>2167</u> Examiner # <u>23933</u>	Where have you searched so far?
Room # <u>3D18</u> Phone <u>23924</u>	USP DWPI EPO JPO ACM IBM TDB
Serial # <u>10/627191</u>	IEEE INSPEC SPI Other <u>EAST & Internet</u>

Is this a "Fast & Focused" Search Request? (Circle One) YES NO
A "Fast & Focused" Search is completed in 2-3 hours (maximum). The search must be on a very specific topic and meet certain criteria. The criteria are posted in EIC2100 and on the EIC2100 NPL Web Page at <http://ptoweb/patents/stic/stic-tc2100.htm>.

What is the topic, novelty, motivation, utility, or other specific details defining the desired focus of this search? Please include the concepts, synonyms, keywords, acronyms, definitions, strategies, and anything else that helps to describe the topic. Please attach a copy of the abstract, background, brief summary, pertinent claims and any citations of relevant art you have found.

topic - hard ware Inventory for system admin.
~~what~~ desired focus: ~~a null value~~ a range as that of claims 3 & 4
Concepts, keywords, acronyms: Background ~~is~~ extremely pertinent, but not towards claims 2-6.
~~def, citation~~
Examples ranges of Ip addresses to identify computers. Some words: Asset Inventory, administrator hardware, audit, asset tracking, high low range
Patents/PGPUBS currently considering: 2003/0225746 (Braun)
2003/0233287 (Sadler)
2003/0093521 (Schonski)
5826000 (hamilton)
want to find - early version of Belmanage System manual or something like it or articles about it.

STIC Searcher LANCE SEALEY Phone 2/8666
Date picked up 2/6/06 Date Completed 2/6/06



SEARCH PREP TIME: 55 MIN.
TERMINAL TIME: 418 MIN.

CLAIMS

- 1 1. A method of managing a computer information database that contains computer
2 profile data for computers, the method including the steps of:
 - 3 A. determining a tree structure of groups for the computers based on primary
4 grouping criteria and secondary grouping criteria;
 - 5 B. including in a database mapping table fields that correspond to primary
6 grouping criteria and secondary groupings criteria for the computers, and including in the
7 fields in respective table records values for profile data of interest that correspond to the
8 primary groupings and secondary groupings, the table further including in the records
9 information that identifies the respective groups to which the values apply;
 - 10 C. receiving for inclusion in the database computer profile data from a plurality of
11 computers;
 - 12 D. for the profile data from a given computer
 - 13 extracting data that corresponds to the profile data of interest for the pri-
14 mary groupings and the secondary groupings,
 - 15 querying the table to determine if the extracted data correspond to the val-
16 ues that are included in the primary grouping and secondary grouping fields in
17 any of the records in the table, and
 - 18 if the query results in one or more table records that include secondary low
19 values, assigning the computer to the group named in the first record found,
 - 20 if the query results in no records, assigning the computer to a default
21 group, or
 - 22 if the query results in multiple records and there are no corresponding sec-
23 ondary low values in the records, assigning the computer to the group in the last
24 record found; and
 - 25 E. manipulating the data from the database to produce reports that summa-
26 rize the attributes of the computers in the groups, with each report for a given
27 group including therein the attributes of the computers in the groups that are on a
28 sub-tree with the given group as its root

1 2. The method of claim 1 wherein one or more table records includes secondary grouping
2 values set to NULL.

1 3. The method of claim 1 wherein the values associated with the primary groupings are
2 ranges and the step of determining if the extracted data correspond to the values further
3 includes determining if the corresponding extracted data of interest falls within one of the
4 primary grouping ranges.

1 4. The method of claim 3 wherein the values associated with the secondary groupings are
2 ranges and the step of determining if the extracted data correspond to the values further
3 includes determining if the corresponding extracted data of interest falls within one of the
4 secondary grouping ranges.

1 5. The method of claim 1 wherein the step of querying further includes determining if the
2 extracted data corresponds to the primary grouping criteria and a secondary low value of
3 NULL or the empty set.

1 6. The method of claim 1 wherein the primary and secondary grouping criteria are user-
2 specified.

1 7. A method of managing a computer information database that contains computer profile
2 data for computers, the method including the steps of:

3 A. determining a tree structure of groups for the computers based on primary
4 grouping criteria;

5 B. including in a database mapping table fields that correspond to a range of val-
6 ues for profile data of interest corresponding to primary grouping criteria used to include
7 the computers in groups for profile data reporting, and including in the fields in respec-
8 tive table records high and low values for the primary grouping profile data of interest,
9 the table further including in the records information that identifies the respective groups
10 to which the values apply;

GROUPING OF COMPUTERS IN A COMPUTER INFORMATION DATABASE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATION

The present application claims the benefit of U.S. Provisional Patent Application
5 Serial No. 60/460,992, which was filed on April 7, 2003, by Gary H. Newman and James
W. Franklin for a GROUPING OF COMPUTERS IN A COMPUTER INFORMATION
DATABASE SYSTEM, and is hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

Field of the Invention

10 The invention relates generally to systems and methods of managing profile data
for a plurality of personal computers and, more particularly, to systems and methods of
grouping personal computer profile data.

Background Information

15 A computer profile includes computer configuration data, such as data that identi-
fies the computer hardware and installed software. The profile may also include other
information, such as, for example, associated software license information, performance
data, and other user specified data. In a prior system for managing a computer informa-
tion database that contains computer profile data, a profile group managing server man-
20 ages the data according to a tree-structured grouping of the computers. The tree struc-
ture, which is designated by the system administrator, may, for example, follow the or-
ganizational chart of a company, with the top level node, or group, corresponding to the
company and lower level nodes, or groups, corresponding to the various branch offices,

and so forth. In the example, the computers may be grouped according to IP subnets that correspond to the branch offices. The profile group managing server then manipulates the profile data to produce reports that summarize the attributes of the computers at every group level, with the reports for a given group including the sub-tree that has the group as its root. A user can then utilize the summaries that are of interest. In the example, a user in a particular branch office may be interested only in the information for the computers in that office, and thus, use the reports produced for the branch office group level. However, a user in the company head office may be interested in the information for all of the company computers, and thus, use the reports produced at the company group level. One such computer information database management system is the BelManage system (version 6) produced by Belarc, Inc., of Maynard, Massachusetts, which is the Assignee of the current invention.

For certain grouping methods, the prior system uses client software that is configured to explicitly specify the particular groups to which the respective computers are assigned. Thus, client software which is configured for a particular group is installed on each computer that is included in the group. For grouping methods based on Lotus Notes Id, the client software uses the organizational structure of the Lotus Notes e-mail addresses of the primary users. For other grouping methods, the prior system uses client software that is configured for the top level group and a group mapping database table to further map the computers to the various lower-level groups.

The group mapping database table has two fields, namely, a profile value-to-match field that contains values of particular profile data and a group field that identifies the groups into which computers with matching profile data values are to be included. The particular data selected for use as the profile value-to-match depends on possible grouping methods. The grouping method may be based on administrator selected groupings, PC Name, Windows Login, Windows Domain or Workgroup, IP address, and so forth. The profile values-to-match maybe, for example, for groupings based on PC Name or Windows Domain the ProfileName or ComputerDomain, respectively.. Further, the groupings based on IP subnet may use selected higher order bits of the computer's IP

address as the profile value-to-match, and the administrator-selected groupings may use other data included in the profiles as the profile value-to-match.

5 The group profile managing server determines which group a given computer belongs in by extracting the data of interest from the profile data received from the computer, and consulting the group mapping database table. If the extracted data matches any of the profile value-to-match entries in the table records, the group managing server includes the computer in the group listed in the first record found to contain an exact match. If no matching record is found, the system includes the computer in the topmost group.

10 In the prior system the administrator changes the group assignment for selected computers either by re-installing properly configured client software on the respective selected computers or, as appropriate, by changing the applicable records in the group mapping database table. When the computers next send their profile data to the profile group managing server, the server manages the data in accordance with the newly defined groups.

15 While the prior system works well, there is a need for a system with greater flexibility to manage groups in ways that more closely follow the internal organization of a company and/or its computer networks. Accordingly, we have improved the BelManage system as described below.

20

SUMMARY OF THE INVENTION

The improved system manages computer profile data using primary and secondary grouping criteria. The system can thus select particular computers using the primary grouping criteria and then further refine the selection using otherwise unrelated secondary grouping criteria. As an example, the system may use IP addresses as the primary grouping criteria and domain name as the secondary grouping criteria for certain or all of the IP addresses.

The system uses a group mapping database table that includes profile value-to-match fields for both the primary grouping and the secondary grouping criteria. In the

ABSTRACT

A computer information database system manages computer profile data grouping the plurality of computers in groups that are nodes of a tree in accordance with user-specified grouping criteria that are respective values of computer profile data of interest, and manipulating the database data to produce summaries of attributes of the computers in a given group and in the groups in the subtree that has the given group as its root. The grouping criteria may be ranges of values for primary grouping criteria, particular values for primary and secondary grouping criteria, ranges of values for both primary and secondary grouping criteria, and so forth. The system uses a group mapping database table that includes profile value-to-match fields for the primary grouping criteria and, as appropriate, the secondary grouping criteria. To use ranges of values, the group mapping database table includes for each range an associated low limit profile value field and a high limit profile value field. If particular values are used as either or both of the primary and secondary criteria, the corresponding low and high limits may be set to the same values or respective value to match fields may be used instead of the high and low limit fields. If there are no secondary grouping criteria associated with a particular primary grouping range, the corresponding entries for the secondary low and high limit profile values or, as appropriate, the entry for the corresponding value to match field is/are set to NULL. Additional grouping criteria may also be used to further refine the selection of computers for the groups, with additional fields for associated values-to-match or ranges included in the group mapping database table. The values or ranges for any or all of the grouping criteria may be altered and/or the values or ranges for grouping criteria may be added to records in which the corresponding fields were set to NULL to change the way in which the computers are grouped.



SYSTEM MANAGEMENT FOR THE INTERNET /

::Easy ::Automatic ::Accurate ::Com

:: PRODUCTS :: WHITE PAPERS :: FREE DOWNLOAD :: ABOUT BELARC :: CONTACT US ::

[HOME](#) > [PRODUCTS](#) > [BelManage](#)

:: BelManage

[Try BelManage](#) | [White Papers](#) | [Pricing](#) | [GSA Pricing](#) | [Requirements](#) | [Purchase](#) | [BelSecure](#)

BelManage is the IT management system designed for the Internet Age. Belarc's IT Portal architecture allows users to simplify and automate the management of all of their desktops, servers and laptops throughout the world, using a single database and Intranet server. Reports include Security Audits, Software Licenses, Windows Server 2003 and XP upgrades, Change History, Health Status, Trend Logs and more. BelManage **features** include the following:

- **Asset Tracking** with detailed software and hardware reports. Quickly determine costs to upgrade to Windows Server 2003 or XP.
- **Software License Compliance** reports to easily see the number of licenses installed.
- **Security Audits** for Microsoft hotfixes and security updates.*
- **Change History** module to automatically track and report all software and hardware adds and removes.
- **Performance Monitoring** of important parameters such as hard drive S.M.A.R.T. status, network activity, CPU utilization, reboots and more.
- Certified to work with **Cisco's NAC** (Network Admission Control) system. [Click here](#) for details.

[Click here](#) to see **sample** BelManage pages.

[Click here](#) to **try BelManage** and find out why our current customers chose to use Belarc's products.

[Click here](#) to request copies of our **white papers** "PC Management for the Internet Age", "System Management for the new Enterprise environment", and "IT as a Utility - Recommendations for Success".

* To deploy missing Microsoft hotfixes and service packs, consider using Microsoft's free "[Windows Server Update Services](#)" product.

Our **Customers** include:





SYSTEM MANAGEMENT FOR THE INTERNET /

::Easy ::Automatic ::Accurate ::Com

:: PRODUCTS :: WHITE PAPERS :: FREE DOWNLOAD :: ABOUT BELARC :: CONTACT US ::

HOME > PRODUCTS > WHITEPAPERS

:: Belarc White Papers

Belarc currently has the following white papers in circulation:

- **"Automating the FISMA Process"**, describes how automated systems such as BelSecure can help U.S. Federal government agencies comply with the FISMA security process.
- **"IT as a Utility - Recommendations for Success"**, outlines the critical elements to have in place for a managed services contract, from the customer's point of view.
- **"PC Management for the Internet Age"**, describes the advantages of using an Intranet based architecture for PC Management.
- **"Policy Management vs. Vulnerability Scanning"**, offers a comparison between these two, often complementary, ways that IT managers can help secure their systems. Vulnerability scanning products test for known vulnerabilities. Policy management products are pro-active by locking the doors in advance of a possible attack.
- **"Security Within™ - Configuration based Security"**, describes the reasons for a configuration-based monitoring system in addition to your existing perimeter security systems, such as anti-virus and firewalls.
- **"System Management for the new Enterprise environment"**, outlines the case of an IT Portal architecture to meet the new requirements placed on today's Enterprise IT management. These requirements include IT security; corporate governance and compliance (Sarbanes-Oxley, FISMA, HIPAA, FFIEC); managing major outsourcing contracts; and working as a team with a geographically distributed organization.

And the following product papers:

- **"Belarc & Cisco NAC"** describes how our products help to automate the use and operation of Cisco's Network Admission Control system.
- **"Self-Assessment Module for IT Security (SAMS)"**. SAMS is based on the U.S. Federal government's FISMA security process and incorporates the NIST Security Self-Assessment Guide for Information Technology Systems (SP 800-26) and the Recommended Security Controls for Federal Information Systems (SP 800-53).

Please send us an [email](mailto:whitepapers@belarc.com) with the title of the white paper(s) you wish to receive, the reason for your interest in IT system management, and where you learned about Belarc: whitepapers@belarc.com

Our **Customers** include:



 For additional information, please contact us at:




Belarc, Inc.
Tel: 978-461-1100
Fax: 509-277-0391
Email: info@belarc.com

Copyright © 2002-5 Belarc, Inc. All rights reserved.

Legal Notice - U.S. Patents 6085229, 5665951 and Patents pending

Freeform Search

Database:	US Pre-Grant Publication Full-Text Database
	US Patents Full-Text Database
	US OCR Full-Text Database
	EPO Abstracts Database
	JPO Abstracts Database
	Derwent World Patents Index
	IBM Technical Disclosure Bulletins

Term:	belarc\$.as.	  
--------------	--------------	---

Display:	<input type="text" value="20"/>	Documents in Display Format:	<input type="text" value="-"/>	Starting with Number	<input type="text" value="1"/>
-----------------	---------------------------------	-------------------------------------	--------------------------------	-----------------------------	--------------------------------

Generate: ☐ Hit List ☒ Hit Count ☐ Side by Side ☐ Image

Search

Clear

Interrupt

Search History

DATE: Monday, February 06, 2006 [Printable Copy](#) [Create Case](#)

<u>Set Name</u> side by side	<u>Query</u>	<u>Hit Count</u>	<u>Set Name</u> result set
<i>DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR</i>			
<u>L2</u>	belarc\$.as.	6	<u>L2</u>
<i>DB=PGPB,USPT; PLUR=YES; OP=OR</i>			
<u>L1</u>	belarc\$.as.	1	<u>L1</u>

END OF SEARCH HISTORY

PATENTS^{IN WHICH} BELMANAGE
PRODUCT IS SUPPOSEDLY
BASED

Set	Items	Description
S1	12	AU=((NEWMAN G? OR NEWMAN, G?) AND (FRANKLIN, J? OR FRANKLIN J?))
S2	4332425	COMPUTER? ? OR CONFIGURAT??? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ? OR ASSET? ? OR AUTOCONFIGURAT???
S3	3247974	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDENTIFI??????
S4	2055645	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BANK? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABANK? ? OR TABLE? ? OR FILE? ?
S5	7	(AU=(NEWMAN G? OR NEWMAN, G? OR FRANKLIN, J? OR FRANKLIN J-?) AND ((S2 OR IT OR IS OR I()T OR I()S)(3N)S3(3N)S4)) NOT (P-D=20020407:20060206)

? show files

File 347:JAPIO Nov 1976-2005/Oct(Updated 060203)

(c) 2006 JPO & JAPIO

File 348:EUROPEAN PATENTS 1978-2006/Jan W04

(c) 2006 European Patent Office

File 349:PCT FULLTEXT 1979-2006/UB=20060105,UT=20051229

(c) 2006 WIPO/Univentio

File 350:Derwent WPIX 1963-2006/UD,UM &UP=200608

(c) 2006 Thomson Derwent

1/5/8 (Item 2 from file: 350)
DIALOG(R)File 350:Derwent WPIX
(c) 2006 Thomson Derwent. All rts. reserv.

016695849 **Image available**
WPI Acc No: 2005-020128/200502
XRPX Acc No: N05-017105

Computer information database managing method, involves determining if
extracted data correspond to values in groupings, and manipulating data
to produce reports for summarizing attributes of computers in groups

Patent Assignee: FRANKLIN J W (FRAN-I); NEWMAN G H (NEWM-I)

Inventor: FRANKLIN J W ; NEWMAN G H

Number of Countries: 001 Number of Patents: 001

Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
US 20040236728	A1	20041125	US 2003460992	P	20030407	200502 B
			US 2003627191	A	20030725	

Priority Applications (No Type Date): US 2003460992 P 20030407; US
2003627191 A 20030725

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 20040236728	A1		11	G06F-007/00	Provisional application US 2003460992

Abstract (Basic): US 20040236728 A1

NOVELTY - The method involves including database mapping table
fields in a database. Data that correspond to a profile data of
interest for primary and secondary groupings are extracted. The
extracted data are queried to determine if the data correspond to
values in the groupings. The data are manipulated to produce reports
for summarizing attributes of the computers in the groups.

USE - Used for managing a computer information database in a number
of personal computers.

ADVANTAGE - The method provides greater flexibility to manage
groups in ways that more closely follow the internal organization of a
company and/or its computer networks.

DESCRIPTION OF DRAWING(S) - The drawing shows a functional block
diagram of a system for managing computer information database.

Computers (10)

Server (14)

Intranet (16)

Computer information database (18)

Profile group management software (20)

pp; 11 DwgNo 1/6

Title Terms: COMPUTER; INFORMATION; DATABASE; MANAGE; METHOD; DETERMINE;
EXTRACT; DATA; CORRESPOND; VALUE; GROUP; MANIPULATE; DATA; PRODUCE;
REPORT; SUMMARY; ATTRIBUTE; COMPUTER; GROUP

Derwent Class: T01

International Patent Class (Main): G06F-007/00

File Segment: EPI

Set	Items	Description
S1	18254551	COMPUTER? ? OR CONFIGURAT??? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ? OR ASSET? ? OR AUTOCONFIGURAT???
S2	10869751	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDE- NTIFI??????
S3	3910972	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BAN- K? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABANK? ? OR TABLE? ? OR FILE? ?
S4	2	AU=(NEWMAN G? OR NEWMAN, G? OR FRANKLIN, J? OR FRANKLIN J?) AND ((S1 OR IT OR IS OR I())T OR I())S) (3N)S2(3N)S3) AND (PD<2- 0020407 OR PY<2003)

? show files

File 2:INSPEC 1898-2006/Jan W3
(c) 2006 Institution of Electrical Engineers

File 6:NTIS 1964-2006/Jan W5
(c) 2006 NTIS, Intl Cpyrght All Rights Res

File 8:Ei Compendex(R) 1970-2006/Jan W5
(c) 2006 Elsevier Eng. Info. Inc.

File 34:SciSearch(R) Cited Ref Sci 1990-2006/Jan W5
(c) 2006 Inst for Sci Info

File 65:Inside Conferences 1993-2006/Jan W5
(c) 2006 BLDSC all rts. reserv.

File 94:JICST-EPlus 1985-2006/Nov W4
(c)2006 Japan Science and Tech Corp(JST)

File 99:Wilson Appl. Sci & Tech Abs 1983-2005/Dec
(c) 2006 The HW Wilson Co.

File 148:Gale Group Trade & Industry DB 1976-2006/Feb 03
(c)2006 The Gale Group

File 636:Gale Group Newsletter DB(TM) 1987-2006/Feb 03
(c) 2006 The Gale Group

Set	Items	Description
S1	535368	CONFIGURAT??? OR AUTOCONFIGURAT??? OR TOPOLOGY OR TOPOLOGI- ES OR LAYOUT? ?
S2	1154363	COMPUTER? ? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?
S3	868643	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDE- NTIFI??????
S4	1335985	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BAN- K? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABAN- K? ? OR TABLE? ? OR FILE? ?
S5	2049787	SEARCH??? OR QUERY??? OR QUERI?? OR SCAN OR SCANNING OR SC- ANNED OR LOOKUP OR LOOK()UP OR RETRIEV??? OR LOCAT??? OR MAT- CH???
S6	1877181	SECOND??? OR OTHER OR ANOTHER OR ADDITIONAL OR EXTRA OR TWO OR 2ND
S7	1741548	GROUP? ? OR TYPE? ? OR RANGE? ? OR ARRAY OR COLLECTION OR CLASSIFIC????? OR DESCRIPTOR? ? OR CATEGOR??? OR SUBTYPE? ? OR MODEL? ? OR CLASS? ?
S8	1090	S5(3N) (S1 OR S2) (3N) S3(3N) S4(3N) S6(3N) S7
S9	233	S8 AND IC=(G06F-007/00 OR G06F-017/30)
S10	112	S9 NOT (PD=20020407:20060206)
S11	0	(S9 AND ((S1 OR S2) (3N) S3(3N) S4)/TI) NOT S10
S12	2148	((S5(100N) (S1 OR S2) (100N) S3(100N) S4(100N) (S6(3N) (S7 OR CR- ITERIA OR CRITERION))) AND IC=(G06F-007/00 OR G06F-017/30)) N- OT (PD=20020407:20060206 OR S10)
S13	8	S12 AND ((S1 OR S2) AND S3 AND S4)/TI
S14	1373	((S5(100N) (S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?) (100N) (INVENTORY OR INVENTORIES OR - TRACK??? OR IDENTIFI??????) (100N) S4(100N) (S6(3N) (S7 OR CRITER- IA OR CRITERION))) AND IC=(G06F-007/00 OR G06F-017/30)) NOT (- PD=20020407:2
S15	20	S14 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR - TECHNOLOGY OR HARDWARE? ?) AND S4)/TI

? show files

File 348:EUROPEAN PATENTS 1978-2006/Jan W04

(c) 2006 European Patent Office

File 349:PCT FULLTEXT 1979-2006/UB=20060105,UT=20051229

(c) 2006 WIPO/Univentio

?

FULL TEXT
PATENT

15/3,K/3 (Item 3 from file: 348)
DIALOG(R) File 348:EUROPEAN PATENTS
(c) 2006 European Patent Office. All rts. reserv.

01181944

An intelligent object-oriented configuration database serializer
Intelligente Serialisierer fur objekt-orientierte Konfigurationsdatenbanken
Serialiseur pour bases de donnees de configuration orientees objet

PATENT ASSIGNEE:

SUN MICROSYSTEMS, INC., (1392733), 901 San Antonio Road, Palo Alto,
California 94303, (US), (Applicant designated States: all)

INVENTOR:

Saulpaugh, Thomas E., 6938 Bret Harte Drive, San Jose, California 95120,
(US)

Slaughter, Gregory L., 3326 Emerson Street, Palo Alto, California 94306,
(US)

Traversat, Bernard A., 2055 California Street, Apt. 402, San Francisco,
California 94109, (US)

LEGAL REPRESENTATIVE:

Harris, Ian Richard et al (72231), D. Young & Co., 21 New Fetter Lane,
London EC4A 1DA, (GB)

PATENT (CC, No, Kind, Date): EP 1030250 A1 000823 (Basic)

APPLICATION (CC, No, Date): EP 301149 000215;

PRIORITY (CC, No, Date): US 253840 990219

DESIGNATED STATES: DE; FR; GB

EXTENDED DESIGNATED STATES: AL; LT; LV; MK; RO; SI

INTERNATIONAL PATENT CLASS (V7): G06F-017/30

ABSTRACT WORD COUNT: 182

NOTE:

Figure number on first page: 3

LANGUAGE (Publication,Procedural,Application): English; English; English
FULLTEXT AVAILABILITY:

Available Text	Language	Update	Word Count
CLAIMS A	(English)	200034	1055
SPEC A	(English)	200034	10768
Total word count - document A			11823
Total word count - document B			0
Total word count - documents A + B			11823

An intelligent object-oriented configuration database serializer
Serialiseur pour bases de donnees de configuration orientees objet
INTERNATIONAL PATENT CLASS (V7): G06F-017/30

...SPECIFICATION THE INVENTION

1. Field of the Invention

This invention relates generally to computer software and **database** systems. More particularly, the invention relates to object-oriented databases and computer languages.

2. Description...

...of database systems includes computerized library systems, automated teller machines, flight reservation systems, computerized parts **inventory** systems, and configuration databases for computer **systems** and **networks**.

Nevertheless, database **systems** are often difficult to maintain. Relational databases, for example, though powerful, are often accessible only...

...administer, and that the data be easy to enter, retrieve, edit, and store.

Some database **systems** are implemented using object-oriented techniques. Object-oriented databases, like the object-oriented programming model...

...One specific type of database is a database employed by an operating system to maintain **configuration** information that relates to components of software and/or **hardware** of a computer system. For example, such a **configuration** database may store **configuration** information relating to application programs, **hardware** devices which are coupled to the computer system, and/or elements of the operating system. These **configuration** databases may be implemented in many different ways. To exploit the advantages of the object-oriented paradigm, **configuration** databases may be implemented as object-oriented databases. Unfortunately, these **configuration** databases, object-oriented or otherwise, are associated with the same difficulties as other types of database **systems**. For instance, if information in a **configuration** database is generated dynamically upon the start-up of a computer system, then that information ...

...complex data types.

In one embodiment, the object-oriented database is an object-oriented **configuration database** which stores configuration parameters pertaining to the software and hardware of a computer system, such...



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) EP 1 030 250 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
23.08.2000 Bulletin 2000/34

(51) Int. Cl.⁷: G06F 17/30

(21) Application number: 00301149.1

(22) Date of filing: 15.02.2000

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 19.02.1999 US 253840

(71) Applicant:
SUN MICROSYSTEMS, INC.
Palo Alto, California 94303 (US)

(72) Inventors:
• Saulpaugh, Thomas E.
San Jose, California 95120 (US)
• Slaughter, Gregory L.
Palo Alto, California 94306 (US)
• Traversat, Bernard A.
San Francisco, California 94109 (US)

(74) Representative:
Harris, Ian Richard et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) An intelligent object-oriented configuration database serializer

(57) A method and system for serializing a transient object-oriented database into a persistent form. The persistent form is a grammatical form, an expression of an object-oriented database in a textual form according to a grammar. The grammatical form is human-readable and human-editable. The grammar is designed to be platform-independent and programming-language-independent and therefore descriptive of any hierarchical object-oriented database. An object-oriented database is expressed as a plurality of entries in a transient, hierarchical, object-oriented form. The free of entries is navigated and each entry is written to the persistent form as text according to the grammar. The serialized form stores only the key state of the database, not a "snapshot" of memory. Therefore, the persistent serialized form is smaller than the in-memory, transient form of the object-oriented database. The object-oriented database is an object-oriented configuration database which stores configuration parameters pertaining to the software and hardware of a computer system, such as application programs, device drivers, system services, and other components. The object-oriented database is platform-independent and is therefore configured to be hosted on several different operating systems and computing platforms.

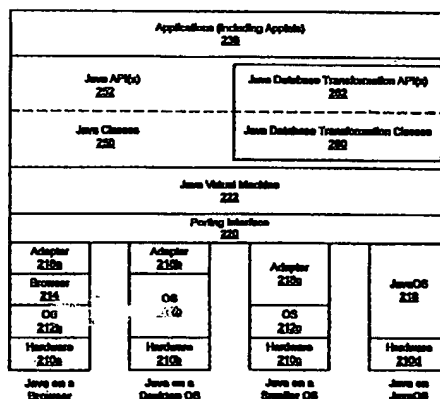


FIG. 3

Description**BACKGROUND OF THE INVENTION****1. Field of the Invention**

[0001] This invention relates generally to computer software and database systems. More particularly, the invention relates to object-oriented databases and computer languages.

2. Description of the Related Art

[0002] Database systems are serving increasingly important roles in today's society. Modern database systems enable users to gather, manipulate, and maintain massive amounts of information. A mere handful of examples of the myriad uses of database systems includes computerized library systems, automated teller machines, flight reservation systems, computerized parts inventory systems, and configuration databases for computer systems and networks.

[0003] Nevertheless, database systems are often difficult to maintain. Relational databases, for example, though powerful, are often accessible only through complicated, formal queries in languages such as SQL (Structured Query Language). It is expensive to hire or train experts with proficiency in such a highly technical field. Storage is also a problem, as data files in a database can become large and unwieldy, consuming finite storage resources. It is therefore an important consideration that a database system be easy to administer, and that the data be easy to enter, retrieve, edit, and store.

[0004] Some database systems are implemented using object-oriented techniques. Object-oriented databases, like the object-oriented programming model, are based on objects: units that combine or encapsulate both data and related methods for operating on that data. Often, objects are related to one another in a class hierarchy which allows related objects to inherit attributes from one another. Object-oriented databases thus provide more accurate modeling of "real-world" entities. However, object-oriented databases are often just as difficult to implement, employ, and maintain as other types of databases. Furthermore, the interdependencies and relationships among objects in an object-oriented database complicate the issue of storage and often result in large, bloated database files which store unnecessary information.

[0005] One specific type of database is a database employed by an operating system to maintain configuration information that relates to components of software and/or hardware of a computer system. For example, such a configuration database may store configuration information relating to application programs, hardware devices which are coupled to the computer system, and/or elements of the operating system. These configuration databases may be implemented in many different ways. To exploit the advantages of the object-oriented paradigm, configuration databases may be implemented as object-oriented databases. Unfortunately, these configuration databases, object-oriented or otherwise, are associated with the same difficulties as other types of database systems. For instance, if information in a configuration database is generated dynamically upon the start-up of a computer system, then that information will be lost from session to session unless it is stored in a convenient way.

[0006] Therefore, it is desirable to provide an intelligent mechanism and process for storing an object-oriented configuration database.

SUMMARY OF THE INVENTION

[0007] Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Features of the dependent claims may be combined with those of the independent claims as appropriate and in combinations other than those explicitly set out in the claims.

[0008] The problems outlined above are in large part solved by various embodiments of a method and system for serializing a transient object-oriented database into a persistent form in accordance with the present invention. Serialization is the process of transforming one or more objects from a transient form to a persistent form. The transient elements of a database do not survive across runtime sessions. A persistent form produced through serialization, on the other hand, can be stored in nonvolatile memory.

[0009] In one embodiment, the persistent form is a grammatical form. A grammatical form is an expression of an object-oriented database in a textual form according to a grammar. The grammatical form is human-readable and human-editable. The grammatical form can be created by hand, or it can be created from an object-oriented database in transient form through the process of serialization. The grammar is designed to be platform-independent and programming-language-independent and therefore descriptive of any hierarchical object-oriented database. In one embodiment, the grammar comprises a set of keywords and a syntax.

[0010] In one embodiment, serialization requires that an object-oriented database be expressed as a plurality of

entries in a transient, hierarchical, object-oriented form. The serializer navigates the tree of entries, writing each entry to the persistent form as text according to the grammar. In this way the textual, persistent form is generated. The persistent form can then be stored in one or more containers. The serialized form stores only the key state of the database, not a "snapshot" of memory as is the default in one embodiment. Therefore, the persistent, serialized form is smaller than the in-memory, transient form of the object-oriented database.

[0011] In various embodiments, the invention further provides a database transformation system and method wherein a persistent form which is described by a grammar is compiled into an intermediate form, wherein the intermediate form populates the active object-oriented database, and wherein serialization and compilation may be modified to accept complex data types.

[0012] In one embodiment, the object-oriented database is an object-oriented configuration database which stores configuration parameters pertaining to the software and hardware of a computer system, such as application programs, device drivers, system services, and other components. In one embodiment the object-oriented database is a platform-independent one, such as the Java™ System Database, and is therefore configured to be hosted on several different operating systems and computing platforms. In one embodiment, database transformation according to the present invention is implemented as a package of classes and interfaces in the object-oriented Java™ Language.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

Fig. 1 is an illustration of a computer system in one embodiment

Fig. 2 is an illustration of the Java™ Platform and the relationships between the elements thereof in one embodiment.

Fig. 3 is an illustration of the Java™ Platform including Java™ Database Transformation functionality in one embodiment of the invention.

Fig. 4 is an illustration of a default hierarchy of the Java™ System Database in one embodiment of the invention.

Fig. 5 is a block diagram illustrating an overview of the transformation of an object-oriented database to and from a grammatical form in one embodiment of the invention.

Fig. 6 is a flowchart illustrating serialization in one embodiment of the invention.

Fig. 7 is an illustration of the correspondence between a grammatical form and an object-oriented form of a database in one embodiment of the invention.

Fig. 8 is an illustration of property domains and attribute domains in the grammar provided by one embodiment of the invention.

Fig. 9 is a further illustration of property domains and attribute domains in the grammar provided by one embodiment of the invention.

Fig. 10 is a flowchart illustrating compilation in one embodiment of the invention.

Fig. 11 is a diagram illustrating nested blocks within the compilation method in one embodiment of the invention.

[0014] While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0015] Turning now to the drawings, Fig. 1 is an illustration of a typical, general-purpose computer system 100

which is suitable for implementing database transformation in accordance with the present invention. The computer system 100 includes at least one central processing unit (CPU) or processor 102. The CPU 102 is coupled to a memory 104 and a read-only memory (ROM) 106. The memory 104 is representative of various types of possible memory: for example, hard disk storage, floppy disk storage, removable disk storage, or random access memory (RAM). As shown in Fig. 1, typically the memory 104 permits two-way access: it is readable and writable. The ROM 106, on the other hand, is readable but not writable. The memory 104 and/or ROM 106 may store instructions and/or data which implement all or part of the database transformation system and method described in detail below, and the memory 104 and/or ROM 106 may be utilized to install the instructions and/or data. In various embodiments, the computer system 100 may comprise a desktop computer, a laptop computer, a palmtop computer, a network computer, a personal digital assistant (PDA), an embedded device, a smart phone, or any other computing device which may exist now or which may be developed in the future.

[0016] The CPU 102 may be coupled to a network 108. The network 108 is representative of various types of possible networks: for example, a local area network (LAN), wide area network (WAN), or the Internet. Database transformation in accordance with the present invention may therefore be implemented on a plurality of heterogeneous or homogeneous networked computer systems 100 through one or more networks 108. The CPU 102 may acquire instructions and/or data for implementing database transformation in accordance with the present invention over the network 108.

[0017] Through an input/output bus 110, the CPU 102 may also be coupled to one or more input/output devices that may include, but are not limited to, video monitors or other displays, track balls, mice, keyboards, microphones, touch-sensitive displays, magnetic or paper tape readers, tablets, styluses, voice recognizers, handwriting recognizers, printers, plotters, scanners, and any other devices for input and/or output. The CPU 102 may acquire instructions and/or data for implementing database transformation in accordance with the present invention through the input/output bus 110.

[0018] In implementing database transformation, the computer system 100 executes one or more computer programs. The computer programs may comprise operating system or other system software, application software, utility software, Java™ applets, and/or any other sequence of instructions. An operating system performs basic tasks such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers. The operating system or other system software may also include a Java™ System Database (JSD) on particular Java™-enabled computer systems, as will be described in detail below. Application software runs on top of the operating system and provides additional functionality. Because applications take advantage of services offered by operating systems, and because operating systems differ in the services they offer and in the way they offer the services, an application must usually be designed to run on a particular operating system. The computer programs are stored in a memory medium or storage medium such as the memory 104 and/or ROM 106, or they may be provided to the CPU 102 through the network 108 or I/O bus 110.

[0019] As will be described in further detail below, the computer system 100 implements a system and method for serializing an object-oriented configuration database into a persistent textual form according to a grammar. In various embodiments, the computer system 100 further implements a database transformation system and method wherein the persistent, grammatical form is compiled into an intelligent intermediate form, wherein the intermediate form populates the active object-oriented database, and wherein serialization and compilation may be modified to accept complex data types. The persistent form may comprise one or more containers. Containers may reside in various forms in the memory 104 on one or more computer systems 100. The database transformation processes may also be referred to as the pushing and pulling of content to and from containers. The pushing and pulling may take place to and from the memory 104, over the network 108, and/or over the I/O bus 110.

[0020] As used herein, an object-oriented database is a database, database system, database management system, electronic filing system, or other computerized collection of information which stores items of data as objects. An object typically includes a collection of data along with methods for manipulating that data. In one embodiment, the database is a configuration database. As used herein, a configuration database is a database, database system, database management system, electronic filing system, or other computerized collection of information which stores information relating to the components and/or parameters which characterize a computer system or systems.

[0021] In one embodiment, the database is implemented using the resources of the object-oriented Java™ Platform and the object-oriented Java™ Language. Furthermore, database transformation is provided via one or more programming interfaces, also known as application programming interfaces or APIs. As used herein, an API is a set of routines, protocols, methods, variables, tools, "building blocks," and/or other resources for building software applications. Therefore, a single API or package of APIs can actually comprise a plurality of APIs of lesser scope. In one embodiment, the database transformation APIs comprise object-oriented interfaces and classes developed in the Java™ Language. These database transformation APIs provide database transformation in accordance with the present invention to Java™ applications which utilize the APIs as "building blocks."

[0022] The Java™ Language is described in The Java Language Specification by Gosling, Joy, and Steele (Addison-Wesley, ISBN 0-201-63451-1), which is incorporated herein by reference. The Java™ Language is an object-ori-

ented programming language. In an object-oriented programming language, data and related methods can be grouped together or encapsulated to form an entity known as an object. The object is the fundamental building block of object-oriented programming. The data structures within an object may alternately be referred to as the object's state, its attributes, its fields, or its variables. In the Java™ Language, the data structures are normally referred to as the variables of the object. If the object represents a telephone, the variables may include a telephone number, a color and a type (e.g., touch-tone or pulse). The procedures which operate on the variables are referred to in Java™ as the methods of the object. In the telephone example, the methods could include ringing, receiving a call or placing a call. These methods will be discussed in more detail below. The variables and methods of an object may all be referred to as the members of the object.

[0023] In object-oriented programming, the grouping together of the variables and methods within an object is referred to as encapsulation. When the variables relating to an object and the methods which might affect the object are encapsulated within the object, other entities usually do not have direct access to these data and procedures. The other entities instead call on the object itself to invoke its own methods and thereby operate on its own data. The encapsulation of the members of the object thereby provides some protection for the data within the object and prevents unauthorized, unwanted, or unintended manipulation of the data. This is sometimes referred to as data hiding. (The concept of data hiding through encapsulation should be distinguished from the hiding of variables in Java™ variable declarations, as explained in more detail below.)

[0024] If a user wants to hide the data within an object, the variable which contains the data is made private. Private variables within an object may only be accessed by the methods of the object. Because it may, in some cases, be inconvenient or impractical to require manipulation of certain data through the methods of the associated object, some variables may be made public. These public variables are directly accessible to entities other than the object with which the variables are associated. Thus, in practice, the variables within objects normally comprise some which are hidden or inaccessible and some which are public.

[0025] All objects in an object-oriented programming system belong to a class, which can be thought of as a category of like objects which describes the characteristics of those objects. Each object is created as an instance of the class by a program. The objects may therefore be said to have been instantiated from the class. The class sets out variables and methods for objects which belong to that class. The definition of the class does not itself create any objects. The class may define initial values for its variables, and it normally defines the methods associated with the class (i.e., includes the program code which is executed when a method is invoked.) The class may thereby provide all of the program code which will be used by objects in the class, hence maximizing re-use of code which is shared by objects in the class.

[0026] Classes in the Java™ Language may be hierarchical. That is, some classes may be subclasses of a higher class, also known as a superclass. For example, in addition to the telephone class (i.e., superclass) above, subclasses may be created for mobile phones and speaker phones. An object which is instantiated from the mobile phone class will also be an object within the telephone class. It may therefore be treated as belonging to the narrower class of only mobile phones, or it may be treated as belonging to the broader class of telephones in general. In the Java™ Language, the subclass (e.g., mobile phones) is said to extend the superclass (e.g., telephones). Alternatively, the superclass is said to be extended by the subclass. For the purposes of this disclosure, a subclass is considered to extend all or any of the classes which are above it in the hierarchy. It may also be said that the subclass directly extends the class immediately above it in the hierarchy, and indirectly extends higher classes in the hierarchy. For example, if a parent class is extended by a first subclass and that subclass is in turn extended by a second subclass, the second subclass can be said to extend the parent class as well as the first subclass. This terminology will also be applied to the hierarchical structure of interfaces, which will be described in more detail below.

[0027] This hierarchical definition of classes and subclasses based on shared variables and methods is very useful. A subclass includes all the variables and methods in the class of which it is a member (its parent class). The subclass is said to inherit the variables and methods of its parent class. This property is useful in defining subclasses because only those variables and methods which do not appear in the parent class need to be defined in the subclass (although variables or methods which appear in the parent class may be redefined in the subclass.) This allows the code written in the parent classes to be reused so that the programmer does not have to rewrite or cut and paste code into each new subclass. Methods that are defined in the parent class may, however, be redefined in subclasses. This is referred to as overriding or hiding the previously defined method(s). By redefining a variable which has already been defined in a superclass, the programmer may hide the previously defined variable (which is distinct from the object-oriented data-hiding concept inherent in encapsulation). In some object-oriented languages, subclasses may inherit variables and methods from several classes. This is called multiple inheritance. If a subclass can only inherit from one parent class, this is called single inheritance. The Java™ Language is characterized by single inheritance, not multiple inheritance.

[0028] This hierarchical class structure also allows the programmer to take advantage of a property referred to as polymorphism. Polymorphism is a mechanism by which various objects may be handled in the same way externally, even though there are differences in the way they are handled internally. In other words, the interface which the different

objects present to an external entity is the same for each object, but the details of each objects implementation may vary. This allows objects instantiated from different subclasses to be handled identically even though the subclasses are not identical. For example, assume that a drawing program implements a class for shapes, a subclass for circles, and a subclass for squares, each of which has a method called draw(). While draw() will be implemented differently for the circle subclass and the square subclass, the drawing program does not have to know the details of how a shape will be drawn, or even which of the shapes is to be drawn. The drawing program simply calls the draw() method for the object to be drawn and the implementation defined in the objects class will be used.

[0029] Another important element of the Java™ Language is the interface. Interfaces are closely related to classes. Interfaces may declare what classes do, but not how they do it. For example, in the case of the telephone class above, an interface would declare that a telephone could ring, place calls, and receive calls, but it would not define the way in which this was accomplished. A telephone class, on the other hand, would set out the functions that define each of these actions so that when a telephone is instantiated, it can actually ring, place a call, or receive a call (in the context of the application). An interface may declare methods and/or constants. To utilize an interface, one or more classes must implement the interface.

[0030] The Java™ Platform which utilizes the object-oriented Java™ Language is a software platform for delivering and running the same applications on a plurality of different operating systems and hardware platforms. As will be described in further detail below, the Java™ Platform includes system-dependent portions and system-independent portions, and therefore the Java™ Platform may be thought of as having multiple embodiments. The Java™ Platform sits on top of these other platforms, in a layer of software above the operating system and above the hardware. Fig. 2 is an illustration of the Java™ Platform and the relationships between the elements thereof in one embodiment. The Java™ Platform has two basic parts: the Java™ Virtual Machine 222, and the Java™ Application Programming Interface (Java™ API). The Java™ API may be thought of as comprising multiple application programming interfaces (APIs). While each underlying platform has its own implementation of the Java™ Virtual Machine 222, there is only one Virtual Machine specification. The Java™ Virtual Machine specification is described in The Java Virtual Machine Specification by Lindholm and Yellin (Addison-Wesley, ISBN 0-201-63452-X), which is incorporated herein by reference. By allowing the Java™ applications 236 to execute on the same Virtual Machine 222 across many different underlying computing platforms, the Java™ Platform can provide a standard, uniform programming interface which allows Java™ applications 236 to run on any hardware on which the Java™ Platform has been implemented. The Java™ Platform is therefore designed to provide a "write once, run anywhere" capability.

[0031] As used herein, "applications" includes applets as well as traditional desktop programs. Applets are Java™ programs that require a browser such as Netscape Navigator, Microsoft Internet Explorer, or Sun Microsystems Hot-Java to run. A browser is a piece of software that allows a user to locate and display Web pages, often encoded in HyperText Markup Language (HTML) and found on the Internet. Typically, applets are embedded in a Web page, downloaded over the Internet from the server, and run on a client machine. Because of security concerns, however, Java™ applets often do not have full access to system services such as read and write access to a file on disk. All Java™ applications 236 require the Java™ Platform to run.

[0032] Developers use the Java™ Language and Java™ APIs to write source code for Java™-powered applications 236. A developer compiles the source code only once to the Java™ Platform, rather than to the machine language of an underlying system. Java™ programs compile to bytecodes which are machine instructions for the Java™ Virtual Machine 222. A program written in the Java™ Language compiles to a bytecode file which can run wherever the Java™ Platform is present, on any underlying operating system and on any hardware. In other words, the same exact Java™ application can run on any computing platform that is running the Java™ Platform. Therefore, Java™ applications 236 are expressed in one form of machine language and are translated by software in the Java™ Platform to another form of machine language which is executable on a particular underlying computer system.

[0033] The Java™ Virtual Machine 222 is implemented in accordance with a specification for a "soft" computer which can be implemented in software or hardware. As used herein, a "virtual machine" is generally a self-contained operating environment that behaves as if it were a separate computer. As shown in Fig. 2, in one embodiment the Java™ Virtual Machine 222 is implemented in a software layer. The same Java™ Virtual Machine 222 can run on a variety of different computing platforms: for example, on a browser 214 sitting on top of an operating system (OS) 212a on top of hardware 210a; on a desktop operating system 212b on top of hardware 210b; on a smaller operating system 212c on top of hardware 210c; or on the JavaOS operating system 218 on top of hardware 210d. Computer hardware 210a, 210b, 210c, and 210d may comprise different hardware platforms. JavaOS 218 is an operating system that is optimized to run on a variety of computing and consumer platforms. The JavaOS 218 operating environment provides a runtime specifically tuned to run applications written in the Java™ Language directly on computer hardware without requiring another operating system.

[0034] The Java™ API or APIs form a standard interface to Java™ applications 236, regardless of the underlying operating system or hardware. The Java™ API or APIs specify a set of programming interfaces between Java™ applications 236 and the Java™ Virtual Machine 222. The Java™ Base API 226 provides the basic language, utility, I/O, net-

work, GUI, and applet services. The Java™ Base API 226 is typically present anywhere the Java™ Platform is present. The Java™ Base Classes 224 are the implementation of the Java™ Base API 226. The Java™ Standard Extension API 230 provides additional capabilities beyond the Java™ Base API 226. The Java™ Standard Extension Classes 228 are the implementation of the Java™ Standard Extension API 230. Other APIs in addition to the Java™ Base API 226 and Java™ Standard Extension API 230 can be provided by the application or underlying operating system. A particular Java™ environment may include additional APIs 234 and the classes 232 which implement them. Each API is organized by groups or sets. Each of the API sets can be implemented as one or more packages or namespaces. Each package groups together a set of classes and interfaces that define a set of related data, constructors, and methods, as is well known in the art of object-oriented programming.

[0035] The porting interface 220 lies below the Java™ Virtual Machine 222 and on top of the different operating systems 212b, 212c, and 218 and browser 214. The porting interface 220 is platform-independent. However, the associated adapters 216a, 216b, and 216c are platform-dependent. The porting interface 220 and adapters 216a, 216b, and 216c enable the Java™ Virtual Machine 222 to be easily ported to new computing platforms without being completely rewritten. The Java™ Virtual Machine 222, the porting interface 220, the adapters 216a, 216b, and 216c, the JavaOS 218, and other similar pieces of software on top of the operating systems 212a, 212b, and 212c may, individually or in combination, act as means for translating the machine language of Java™ applications 236, APIs 226 and 230, and Classes 224 and 228 into a different machine language which is directly executable on the underlying hardware.

[0036] Fig. 3 is an illustration of the Java™ Platform including database transformation functionality in one embodiment of the invention. The Java™ API 252 comprises all of the API sets available to the applications 236: the Java™ Base API 226, optionally the Java™ Standard Extension API 230, and other APIs 234. The Java™ API 252 also encompasses the Java™ database transformation API(s) 262, the programming interfaces between applications 236 and the Java™ Virtual Machine 222 for transforming a database from a run-time, object-oriented form to a persistent, grammatical form and back again. The Java™ Classes 250 are the implementation of the Java™ API 252, including the Java™ Base Classes 224, optionally the Java™ Standard Extension Classes 228, and other classes 232. The Java™ database transformation API(s) 262 are implemented by the Java™ database transformation classes 260.

[0037] Therefore, as shown in Fig. 3, the database transformation framework is implemented in a software layer of APIs and classes between the Virtual Machine 222 and the Java™ Applications 236. The APIs and the applications comprise Java™ source code and/or bytecodes which are executable by the Virtual Machine. Therefore, the APIs can be ported without alteration to any underlying computing platform on which the Java™ Virtual Machine has been implemented. Because of the widespread implementation of the Java™ Virtual Machine, the APIs can be implemented with ease on a plurality of operating systems and computer hardware. The database transformation APIs thus enable Java™ applications to implement the system and methods of the present invention in a standardized, cross-platform manner.

[0038] The system, method, and storage medium of the present invention are applicable to any object-oriented database. In one embodiment, the object-oriented configuration database is a Java™ System Database (JSD), also known as a JavaOS System Database. The JSD is platform-independent but was developed in conjunction with JavaOS. In other words, the JSD could be hosted on many different operating systems, including JavaOS. The JSD generally allows an operating system, system services, applications, utilities, and other software components to store and retrieve configuration information concerning the software and hardware of a platform, typically a Java™-based platform such as a network computer (NC). Configuration information is arranged to describe, for example, the physical devices that are present in a machine associated with the JSD, the system software services that are installed, and specific user and group application profiles. The JSD serves as a central repository to store, as well as access, substantially any information which is used for configuration purposes.

[0039] The JSD is comprised of a hierarchy or tree of entries. Entries can represent files, applications, users, devices, public interfaces, and many other components. Each entry has a single parent entry and any number of child entries. An entry has a unique name which describes the location of the entry relative to the root of the tree. The root is identified by a single forward slash ("/"). Each entry is uniquely identified using a UNIX-style pathname, composed of a list of all entries preceding it in the hierarchy, with each component separated by a forward slash ("/"). In addition, an entry may also contain zero or more associated properties and/or attributes. A property is a user-defined piece of information which consists of a name-value pair. This information may be any Java™ object. An attribute is system-defined data or meta-data and is generally not available to non-system components. An attribute may be an entry attribute which is associated with the entry itself, or an attribute may be a property attribute which is associated with a specific property of an entry.

[0040] A large amount of information in the JSD is transient: it does not survive across runtime sessions. The JSD is populated, that is, filled with entries relating to configuration data, during platform initialization. Additional configuration data is added and/or removed when the operating system boots. Transient information must be repopulated from its source into the JSD every time the computer system or operating system boots. Every time JavaOS boots, for example, the JSD is repopulated with information concerning which devices are installed on the platform. The JSD provides a population interface in the Java™ Language for adding configuration data from a variety of sources: for example, files,

a network, the host operating system, applications, and drivers. In one embodiment, the Java™ Language interface for JSD population is named TreePopulator.

[0041] The JSD uses a split design: one part resides on a server computer system, and the other part resides on a client computer system. On the server, the configuration information is stored for each user and client machine on the network. At the time the client computer system boots, each client database is populated from the server database with configuration information about a particular machine, group of machines, and machine platform. At the time a user logs in, the client database is populated with configuration information about the user who is logging in and the group of users he or she belongs to, if any.

[0042] Fig. 4 illustrates the hierarchical nature of the JSD. In one embodiment, the JSD is divided into six standard namespaces, or sub-trees of related entries, which are created when JavaOS starts: Temp, Device, Interface, Alias, Software, and Config. Entries within a given namespace share common characteristics. A default namespace manager manages each namespace, controlling how entries are created, added, accessed, removed, and updated for a particular namespace. When an entry is published (that is, added to the database and thus made public), it inherits its parent's namespace manager by default.

[0043] The Temp namespace is available as temporary storage for both application and system software settings. The Device namespace contains the set of devices available to the local platform. The Interface namespace contains entries that reference services that implement public Java interfaces. The Alias namespace contains entries that reference entries in the Interface namespace and provide friendly naming schemes for existing entries. The Software namespace contains entries for each installed software component. The Config namespace maintains client configuration information and is usually stored on servers.

[0044] In one embodiment, the Temp, Device, Interface, and Alias namespaces are transient: they do not survive across runtime sessions, typically because they are stored in volatile memory and not in nonvolatile or persistent storage. The Config namespace is persistent. The Software namespace is transient on clients but is backed up persistently on the server.

[0045] Fig. 4 is an illustration of a tree structure representing an exemplary Java System Database on a client computer system. The client tree 301 resides on a networked client machine 300 and relates to configuration data of the client computer system 300. The client machine 300 is an example of a computer system 100 as discussed with reference to Fig. 1. The hierarchy of the client tree 301 is manifested using an n-way tree. At the root of the tree is a root entry 302 which does not contain any data. A first level of nodes 304 in client tree 301 collectively define the six standard namespaces as discussed above.

[0046] For example, the Software namespace begins at node 306 and includes all nodes and data branching from node 306. All entries in the Software namespace relate to configuration data regarding software applications for the client computer system 300. Entries in the data schema are made up of a unique name, a list of children (entries below the given entry), and a set of tuples. Each tuple contains a property name and associated property value (i.e., a name-value pair). In a word processing program, for example, a property name can be "font" and the property value can be "Times Roman." Similarly, all entries under the Device namespace 308 are entries that are related to configuration information of the client computer system 300. Every entry in the hierarchy may act as both an entry in a sub-tree and the root of a sub-tree having descendant entries or child nodes. Each namespace in layer 304 is described in U.S. Provisional Application filed on May 14, 1998 and commonly assigned, entitled "JAVA SYSTEM DATABASE," which is incorporated herein by reference.

[0047] The Software namespace 306 contains a list of installed and/or available system services such as device drivers, user applications, and user configuration information. The Software namespace 306 includes four categories: application, system, service, and public. In the application category 312, for example, an entry com.Netscape 314 contains the company-unique name "Netscape." Below com.Netscape 314 is an entry 316 for Netscape Navigator, one of Netscape's products. Below the Navigator entry 316 is an entry 318 storing company-specific configuration information relating to Netscape Navigator. The Netscape Navigator application program could access the configuration information 318 while the application program is executing. In a similar way, other application programs could access their own specific configuration entries while executing. Entries 320, 322, and 324 represent other vendors which will also have application-level entries similar to entry 316.

[0048] An object-oriented database such as the JSD is largely transient and thus must typically be recreated with every runtime session. Furthermore, the JSD may be difficult to access for data entry and retrieval when it is active in the cache as described above. Therefore, one embodiment of the present invention provides for database transformation from the active, run-time form to a more manageable, persistent form, and then back again. Fig. 5 illustrates an overview of database transformation in accordance with one embodiment of the present invention. The object-oriented configuration database 340 can undergo a process of serialization 342 which transforms the database into a persistent form in one or more containers. The persistent form, also known as a grammatical form 346, is described by a grammar 344. The grammatical form 346 can undergo a process of compilation 348 which transforms the persistent form into an intelligent intermediate form 350. The intelligent intermediate form 350 can be turned back into the object-oriented

database 340 through the process of database population 352. A transformation customizer or plug-in 354 can be used to extend the grammar 344 to allow for the use of complex data types in serialization 342 and compilation 348. Although the grammatical form 346 is much smaller than the in-memory object-oriented database 340 due to improved serialization 342, no important information is permanently lost when transforming one into the other, and therefore one form can be transformed into the other form and back again an indefinite number of times.

[0049] The JSD is an in-memory repository or cache which can be stored in persistent containers such as files in accordance with the present invention. The JSD defines a public API for dynamically adding, removing, and modifying entries contained within the database. In one embodiment the public API is a Java™ Language interface named Tree-Populator. For instance, when the grammatical form is parsed by the configuration tree compiler into an intermediate form, the resulting hierarchical entries of the intermediate form are imported into the JSD using the public API. In this way, the content of the JSD is pushed and pulled from the containers. A single JSD can push and pull content from multiple containers as required to satisfy clients. The JSD pushes and pulls content without regard to container count and implementation. The container count and implementation vary with the complexity and needs of the specific platform. For example, a simple platform such as a cellular phone may have a single persistent container stored in non-volatile RAM (NVRAM), while a more complex platform such as a network computer may use multiple file-based containers or even an enterprise network directory service such as LDAP (Lightweight Directory Access Protocol).

[0050] The values and structure of the JSD content remain the same whether active in the cache or persistently stored in a container. However, the format of the content does change when moved to and from a persistent container. When the content is active in the cache, its format (how its values and structure are represented to software) is that of a Java™ object. Cached JSD objects behave in a similar way to other objects in the Java™ Language, except that a JSD object's construction and serialization is automated by the JSD. When the content is stored in a container outside the cache, however, the object's values and structure are represented using the database description grammar (DDG or grammar) in accordance with the present invention.

[0051] As used herein, serialization is any process of transforming one or more objects from a run-time or transient form to a persistent form. The Java™ Language supports a default object serialization that flattens a graph of objects, often hierarchical, into a byte stream. The byte stream can then be stored in one or more persistent containers such as files. Subsequently, the byte stream can be reconstituted into live Java™ Language objects. Nevertheless, the default method takes a complete snapshot of the live object graph, including internal references to other objects. Therefore, the default method is slow and produces a large amount of data to store. For largely the same reasons, and especially because the default serialized form includes references among objects, the default form cannot be edited manually. Improved serialization according to one embodiment of the present invention provides a number of advantages over the default method: faster speed, smaller output size comprising only the key state, a simple object hierarchy, and text that is editable by hand using a text editor. Furthermore, the improved serialization method according to one embodiment of the present invention produces a persistent form that is not dependent on the Java™ Language or Java™ Platform.

[0052] In one embodiment, serialization is implemented in the Java™ Language. The class StaticTreeSerializer contains the code to push database content from the in-memory JSD cache into a grammar-based, persistent container such as a file. An instance of StaticTreeSerializer requires five parameters to be passed to its constructor. The first two parameters are PrintStream instances: an error log and a debugging log. The third construction parameter is a debugging level that controls how much compilation progress information is output to the debug log. The fourth parameter is an instance of a Entry (java.xml.system.data-base.Entry) which is used by the serializer to reference a portion (a tree) of the active database to serialize. In other words, the Entry parameter to the serializer determines which node will be the root of the tree or sub-tree to be serialized. The final construction parameter is a reference to an output file to hold the grammatical form. The output file can be specified with a descriptor (java.io.FileDescriptor), a file name (java.lang.String), a file (java.io.File), or a file output stream (java.io.FileOutputStream).

[0053] Fig. 6 illustrates serialization of a tree in one embodiment. In step 502 the serializer begins by creating a grammar indentation buffer. This buffer contains spaces to indent the grammar for each entry, property, or attribute scope. Next, in step 504 the serializer creates a JSD transaction that locks the target tree of entries for exclusive access. In steps 506 and 508, respectively, the serializer then uses JSD APIs to walk the tree of entries and produce a description of the entries in grammatical, textual form to the output file. In doing so, the serializer maintains only the key state of the objects, such as the hierarchy, names of entries, and name-value pairs for properties and attributes, according to the grammar which is described in detail below. Unlike default Java™ serialization, the serializer of the present invention does not maintain a complete in-memory snapshot of objects, and thus the data produced is much smaller in size than with default serialization. Furthermore, by eliminating Java™-specific object information such as dependencies from one object to another, the textual form is conveniently editable by hand with a text editor program. Steps 506 and 508 continue until the serializer determines in step 510 that the last entry has been reached. After the last entry in the JSD has been processed, the output file is flushed and closed in step 512. Finally, in step 514, the JSD transaction is committed and the serializer exits.

[0054] In one embodiment, the text-based, grammatical description of the contents of the object-oriented database

is a static configuration free. A static configuration free provides a mechanism whereby hierarchies such as those of an object-oriented configuration database are defined and expressed in a persistent medium using a grammar. The configuration free is static because each time the grammar description is compiled, it results in the same hierarchy being created within the JSD. Static configuration information is ideal for devices, applications, and services that are simple in nature and do not require dynamic discovery of devices or resources at boot time or service load time.

[0055] As used herein, a grammar is a set of rules that govern the structure and/or relationships of terms and/or symbols in a language. As used herein, a grammatical form is a collection of information which is expressed under the rules of a particular grammar. The grammar provided by the present invention is independent of any platform or programming language: the grammar could be used to describe any hierarchical, object-oriented database. A small number of keywords and a simple syntax define the static configuration grammar. Generally speaking, a syntax defines a structure in which the keywords and other elements can be expressed. In other words, a syntax generally relates to the form of the grammar. In one embodiment, the keywords are as follows:

TREE *name*: Defines the static free root with the specified name.

ENTRY *name*: Defines a new entry with the specified name.

PROPERTIES: Defines one or more properties for the entry.

ATTRIBUTES: Defines one or more attributes to pertain to the entry if no properties have yet been defined, or otherwise to the last property defined.

BUSINESS_CARD: Defines a collection of configuration information for an application, device driver, or other software component.

[0056] The TREE and ENTRY keywords are used to define the free root and the free sub-hierarchy, respectively. The ATTRIBUTES and PROPERTIES keywords, along with name-value pairs, define attributes and properties to be associated with an entry. The scope of a keyword is delimited using opening and closing braces (i.e., curly brackets) "{" and "}". The TREE keyword must appear first because it defines the root of the configuration tree.

[0057] Fig. 7 illustrates an example of this hierarchy. A configuration tree 620 is defined using the TREE and ENTRY keywords and the scoping braces of the grammar described above:

```

TREE root {
    ENTRY child1 {
        ENTRY grandchild1 {
        }
        ENTRY grandchild2 {
        }
        ENTRY grandchild3 {
        }
    }
    ENTRY child2 {
    }
}

```

[0058] The static configuration tree 620 is different in form but identical in content to a configuration database 600. The root 602 has two children, child1 604 and child2 606. There are three grandchildren 608, 610, and 612, all belonging to child1 604. The tree 620 can be transformed into the database 600 through the processes of compilation and/or database population as described in detail below, and the database 600 can be transformed into the tree 620 through the process of serialization as described in detail above. Although the grammar allows the database to be represented in a much more compact form than the in-memory form of the database, no important information is permanently lost in transforming the database to or from the grammatical form. Therefore, the transformations can take place an indefinite number of times back and forth.

[0059] While the hierarchy discussed with reference to Fig. 7 conveys information regarding the relationships of the entries, true entry customizing is accomplished using properties and attributes. Fig. 8 illustrates the property and attribute domains for a given entry. As shown in Fig. 8, properties pertain only to the entry itself, but attributes can be assigned either to the entry or to any existing property of the entry. The JSD public API provides methods for dealing

with both types of attributes. The name of a property or attribute must be unique within its domain, but only within its domain. For example, as shown in Fig. 8, it is acceptable that the attribute "X" appears in all three defined attribute domains. Each "X" is distinct from the others because their respective scopes are different, and the three may have different values of different types. In one embodiment, properties and attributes are defined by the JSD to have names of the standard Java™ type java.lang.String and values of the standard Java™ type java.lang.Object. In one embodiment, null values are not permitted.

[0060] The PROPERTIES and ATTRIBUTES keywords are used to designate name-value pairs for the purpose of defining properties and attributes, respectively. For both properties and attributes, the general definition syntax is:

name = value;

[0061] In one embodiment, the "name" term must be compatible with java.lang.String and may therefore contain any Unicode character, except for the two-character sequence "/" (a forward slash followed by a period). The semicolon after the "value" term delimits each name-value pair. Using pseudo Backus-Naur Formalism (BNF) as is well-known in the art of computer languages, the value can be defined as:

```

value = [ unicode_string |
        "[ type ":" val "]" |
        "[ type "[:" val 0*[ "," val ] "]"
        ] ","
type = "boolean" | "byte" | "char" | "int" | "long" | "short" | "double" | "float" |
"string"
val = "true" | "false" | unicode_string | dec-val
unicode_string = 0*unicode_char

```

[0062] Furthermore, the value provided must match the specified type. The supported primitive types are: boolean, byte, char, int, long, short, double, float, string, and arrays thereof (boolean[], byte[], char[], int[], long[], short[], double[], float[], string[]). The default type is a Unicode string, which may also be explicitly specified using the string type. Similar to assigning the property or attribute name, a string value may be any series of Unicode characters compatible with java.lang.String. Note that the string value is not enclosed in quotation marks. For string values, the limitation on using the two-character sequence "/" (a forward slash followed by a period) does not apply. When the type specification is followed by the Java™ array specification characters "[]", then the type is an array. In an array, one or more elements must be specified via a comma-separated list of values.

[0063] An example of the designation of values in a hierarchy is as follows, with comments introduced by two forward slashes ("//"):

```

TREE test {
    ENTRY child {
        5      ATTRIBUTES {
                // Entry attribute "AttrName1" is an array of two bytes
                AttrName1 = [byte[:23,42];
            }
        10     PROPERTIES {
                // Assigns string "Hello, World!" to property "PropName1"
                PropName1 = Hello, World!;

                // Property "Claimed" is boolean with a value of true
        15     Claimed = [boolean:true];

                // Property data is an array of four longs
        20     data = [long[:23,87,9009834,345];

                // Property data_names is an array of four strings
                data_names = [string[:width,height,weight,days_left];
        25     }
    }
}

```

30 [0064] Attributes assigned to the entry attribute domain must appear before any properties within the scope of the ENTRY keyword declaration. Thus, in the previous example, the attribute "AttrName1" is assigned to the entry, rather than to a property of the entry. All properties appear only to the entry and thus are always in the entry property domain. Thus, they may appear anywhere within the scope of the ENTRY declaration, except not within the scope of an

35 ATTRIBUTES or another PROPERTIES declaration.

[0065] Attributes pertain to the last defined entry or property. Again, in order to be applied to the entry itself, the ATTRIBUTES declaration must immediately follow the ENTRY line before any properties are defined. Otherwise, the attributes will be assigned to the domain of the last defined property. The following example further illustrates assigning attributes and properties to a specific domain, with comments introduced by two forward slashes ("/"):
 40

```

TREE test {
    ENTRY child {
5         // These attributes are assigned to the entry
        ATTRIBUTES {
            name1 = value1;
            name2 = [integer:1024];
10        }
        PROPERTIES {
            name1 = value1;
            name2 = [boolean:false];
15        // These attributes are assigned to the property "name2"
        ATTRIBUTES {
            name1 = [byte[:4,5,8];
            name2 = [string:I am name2];
20        }
        name3 = [char:H];
        name4 = [boolean[:true,true,false];
        // These attributes are assigned to the property "name4"
        ATTRIBUTES {
25            name1 = attribute one;
            name2 = [integer:777];
            name3 = [char[:a,b,c];
30        }
    }
}

```

[0066] The entry and attribute domains of the previous example are further illustrated by Fig. 9. As described above, the child entry 702 has both a property domain 704 and an attribute domain 706. The entry property domain has four properties, including name2 708 and name4 710. Both name2 708 and name4 710 have attribute domains of their own: name2 attribute domain 712 and name4 attribute domain 714, respectively.

[0067] The database description grammar is language- and platform-independent. Nevertheless, a database description which follows the rules of the grammar can be compiled into an intermediate form and then used to populate the JSD. As used herein, compilation is any process of transforming information expressed in a first language to information expressed in a second language by applying one or more grammars to interpret and/or analyze the first form and create the second form. Usually, although not always for the purpose of this disclosure, compilation is a process of transforming information expressed in a higher-level language into information expressed in a lower-level language. The lower the level of a language, generally speaking, the easier it is for a computer to understand or execute: in other words, the more readable it is by a computer. The higher the level of a language, generally speaking, the more readable it is by a human.

[0068] In one embodiment the compiler is a Java™ class called StaticTreeCompiler. An instance of StaticTreeCompiler requires four parameters to be passed to its constructor. The first two parameters are PrintStream instances: an error log and a debugging log. The third construction parameter is a debugging level that controls how much compilation progress information is output to the debug log. The fourth and final parameter is an instance of a StaticTree used by the compiler to obtain tokens. A token is a single meaningful element of a computer language, such as, for example, a keyword, a name associated with a keyword such as the name of an entry, a name for an attribute or property, a value for an attribute or property, an opening or closing brace to indicate scope, or another punctuation mark. In other words, the StaticTree is a persistent form containing a grammatical form of a database, as described in detail above.

[0069] The constructor stores these configuration parameters and then allocates a large scratch array of StaticEn-

tries in preparation to compile the grammatical form to an intermediate form. The StaticEntry objects, which are instances of a StaticEntry class, represent entries in the intermediate form of the database. A StaticEntry object contains references to its parent siblings, and children, as well as its properties (StaticProperty instances) and attributes (StaticAttribute instances). Furthermore, each StaticEntry contains the entry ID and name. The StaticAttribute, StaticProperty, and StaticEntry classes are defined as follows:

```

class StaticAttribute {
    public String attrName;
    public Object attrValue;

    StaticAttribute (String name, Object value) {
        attrName = name;
        attrValue = value;
    }
}

class StaticProperty {
    public String propName;
    public Object propValue;
    public StaticAttribute[] propAttributes;

    StaticProperty (String name, Object value) {
        propName = name;
        propValue = value;
    }
}

class StaticEntry {
    public StaticEntry entryParent;
    public StaticEntry entrySibling;
    public StaticEntry entryFirstChild;
    public int id;
    public String entryName;
    public StaticProperty[] entryProperties;
    public StaticAttribute[] entryAttributes;

    StaticEntry (StaticEntry parent, int ID, String name, int maxProperties)
    {...}
}

```



```

    public int addProperty(String name, Object value) {...}
    public StaticProperty lastPropertyAdded() {...}
    public int addAttribute(String attrName, Object value) {...}
5    public int addAttribute(StaticProperty p, String attrName, Object value)
        {...}
    public void finalizeProperties() {...}
10 }

```

[0070] As discussed above, each instance of a StaticTreeCompiler has a reference to a StaticTree instance. The StaticTree instance in turn either has a reference to a file or to an array of Java language strings containing the grammatical form that is to be compiled. When the compile() method is invoked, the StaticTree's constructor builds either a stream or string tokenizer to process the grammatical form contained within the file or within the array of java.lang.String. The StaticTree class provides methods to the compiler such as getNextToken() and hasMoreTokens(). The compiler invokes these methods to parse the grammatical form into a series of tokens. The tokens cause the compiler to recognize the grammatical form and create StaticEntry, StaticProperty, and StaticAttribute objects that together comprise the intermediate form of the database. In one embodiment, in other words, the compiler is a recursive descent parser that repeatedly reads tokens from the StaticTree token parser until the grammatical form is fully consumed. The resulting StaticEntries are placed in the large scratch array. When the end of the grammatical form is recognized, a final array of StaticEntries is created that is the exact size as the number of entries defined in the grammatical form. The compiler also squeezes out unused space in each StaticEntry for properties and attributes.

[0071] The compilation method according to one embodiment is described in more detail as follows. The compiler is started by calling or invoking the compile() method, which is illustrated in Fig. 10. In step 750 the compile() method initializes an error condition variable to "kNoErr" and initializes a grammar scope count (indicating depth in the hierarchy) to zero. Also in step 750, a large temporary or "scratch" array of intermediate-form entries is allocated to hold entries which will be produced by the compiler. In step 752 a loop is ten entered that continues until either an error condition is discovered or the StaticTree parser fails to return a token. In step 754 the compiler reads the next token from the grammatical form. In step 756 the compiler decodes the token type. Each JSD tree defined in the grammatical form begins with the special token "TREE". If the token is determined in step 756 to be the beginning token "TREE", then in step 758 the compiler descends into a compileTree() method to compile the tree. The scope of each entry, property, or attribute begins with a "{" token and ends with a "}" token. If the token is determined in step 756 to be the beginning scope token "{", then in step 759 the grammar scope count is incremented. If the token is determined in step 756 to be the ending scope token "}", then in step 760 the grammar scope count is decremented. If the token is determined in step 756 to be "ENTRY", then in step 761 the compiler descends into a compileEntry() method to compile the entry. If the token is determined in step 756 to be "PROPERTIES", then in step 762 the compiler compiles the property or properties. If the token is determined in step 756 to be "ATTRIBUTES", then in step 763 the compiler compiles the attributes. If the token is determined in step 756 to be an unrecognized token, then the compiler searches for the appropriate plug-in to translate or otherwise process the unknown token. In one embodiment, for example, the "BUSINESS_CARD" token is supplied by a plug-in. Plug-ins for transformation customization are discussed in detail below.

[0072] Regardless of the token type, the compiler then loops back to step 752 to continue processing the grammatical form. After all the tokens in the grammatical form have been read and processed, then in step 764 the temporary scratch array of static entries is compressed to the exact size required to hold the intermediate form.

[0073] The compileTree() method has the following logic. A special StaticEntry to mark the tree root is allocated and initialized. Its ID is zero, and the compiler global variable that tracks the next ID to use is initialized to one. As the compiler processes each entry in the grammatical form of the database, an instance of StaticEntry is created and added to the large array of static entries. In doing so, the compileEntry() method is invoked, wherein a loop is entered that completes upon an error condition or upon encountering the end of the tree.

[0074] The compileEntry() method has the following logic. An error condition variable is initialized to "kNoErr" and a grammar scope count is initialized to zero. A loop is then entered that exits upon an error condition, upon the failure to read a token, or when the scope count indicates the end of the current entry's definition. The definition can include child entries and sets of properties and/or attributes. As the compiler processes each property and attribute defined in the grammatical form, instances of StaticProperty and StaticAttribute are added to the current StaticEntry using the addProperty and addAttribute methods, respectively. The current StaticEntry (the last one added to the array) is obtained by the compiler using a compiler instance variable containing the last array index (entry ID) used. The compiler

can use the lastPropertyAdded() method to obtain a StaticProperty object that represents the last property added to the current entry.

[0075] Therefore, the compiler logic can be expressed in pseudo-code as follows:

```

5      public void compile() {
        Initialize scratch array of intermediate-form static entries
        while ( no errors AND tree to compile has tokens) {
10         .   read a token
            .   decode token type
            .   tree begin token?
            .       if YES, compileTree()
            .   scope begin token?
15         .       if YES, increment count
            .   scope end token?
            .       if YES, decrement count
            .   entry token?
20         .
            .
            .
            .
            .   if YES, compileEntry()
25         .   properties token?
            .       if YES, compile property or properties
            .   attributes token?
            .       if YES, compile attribute(s).
30         .   unrecognized token?
            .       if YES, look for appropriate plug-in
            .   }
        Compress scratch array of intermediate-form static entries into final form,
35         optimizing size
    }

```

[0076] Furthermore, the compiler logic can be expressed in nested blocks as shown in Fig. 11. The compile() method 770 has an outermost block 772 which begins by initializing a temporary array of independent-form entries, as explained above. When the "TREE" token is encountered, the compile() method 770 enters a next block or loop 774 by invoking the compileTree() method. For each entry in the tree, the compile() method then enters a block or loop 776 by invoking the compileEntry() method. Within the compileEntry() method 776, in block 778 the compiler compiles the properties, attributes, and child entries as described in the grammatical form. After all the entries have been compiled, the compile() method returns to the outermost block 772, in which the intermediate form of the database is finalized and compressed, as described above.

[0077] The StaticTreeCompiler produces an intermediate form, not a final form, of the database. The intermediate form lacks the infrastructure of the final database; for instance, the intermediate form is not designed to be accessible by particular application programs for storage of their configuration information. The intermediate form is accessed by the StaticTreePopulator which populates the database. Each StaticTreePopulator has an instance of a compiler and each instance of a compiler has a reference to a StaticTree instance. When the populator is constructed, its constructor is passed an instance of a StaticTree to compile. In one embodiment, therefore, compilation and population are both initiated when a StaticTree instance is passed to a StaticTreePopulator.

[0078] During StaticTreePopulator construction, the populator creates an instance of the compiler and then invokes the compile() method as discussed in detail above. The compile() method then creates the intermediate form. In one embodiment, the intermediate form is a simple array of JSD entry objects which are indexed by an entry ID. Each entry

object is represented in the intermediate form as an instance of StaticEntry class. Therefore, the objects of the intermediate form are intelligent: they combine data with methods for manipulating the data. For example, the intermediate form includes methods for creating a database entry, creating a property associated with an entry, creating an attribute associated with an entry or a property, querying the last entry, property, or attribute created, and finalizing entry storage.

[0079] The compiler uses the constructor and utility methods to build up the intermediate form during the compilation process, as described in detail above. The populator directly accesses the array by entry ID. The entry IDs are bounds-checked against the StaticEntry array before the populator accesses the array of static entries. An invalid entry ID will cause the populator to return an invalidEntryIDException to the JSD. This error should never be encountered if all the IDs were generated by the compiler and returned by the populator.

[0080] In one embodiment the process of populating the JSD is encapsulated in a Java™ Language interface called TreePopulator. Software components wishing to populate the database must implement this interface. In other words, developers who wish to create software components that populate the database should implement the TreePopulator interface in their Java™ applications. Those components that do implement the TreePopulator interface are referred to as database populators. The TreePopulator interface is defined as follows:

```

package javaos.javax.system.database;
public interface TreePopulator {
    public int getRootEntry();
    public String getEntryName(int entry);
    public hit getParentEntry(int entry);
    public hit getFirstChildEntry(int entry);
    public jut getPeerEntry(int entry);
    public hit getPropertyLength();
    public String getNextProperty(int entry, String prevPropName) throws SystemDatabaseException;
    public Object getPropertyValue(int entry, String propName) throws SystemDatabaseException;
    public hit getAttributeNameLength();
    public String getNextAttribute(int entry, String prevAttrName) throws SystemDatabaseException;
    public String getNextAttribute(int entry, String propName, String prevAttrName) throws SystemDatabaseException;
    public Object getAttributeValue(int entry, String attrName) throws SystemDatabaseException;
    public Object getAttributeValue(int entry, String propName, String attrName) throws SystemDatabaseException;
}

```

[0081] The TreePopulator interface is used by the JSD to import a set of entries, also referred to as a tree. JSD populators, that is, software components or applications which implement the TreePopulator interface, are not required to use the database description grammar. Like all JSD populators, however, grammar-based populators must implement the TreePopulator interface. The StaticTreePopulator class is the default implementation of a grammar-based populator. The StaticTreePopulator class encapsulates a compiler (StaticTreeCompiler) and a compiled tree of grammatically derived entries. As described above, the compilation takes place during the StaticTreePopulator construction process. If the compilation completes correctly, then the StaticTreePopulator is ready to populate the JSD.

[0082] The StaticTreePopulator understands the intermediate database format. Each invocation of the TreePopulator interface requires an intermediate-form entry lookup, and possibly a property or attribute lookup as well. The results of the lookup (entries, properties, or attributes) are then returned to the JSD for inclusion in the active cache of entries. During population, each level of the tree is read an entry at a time until all entries are returned to the JSD by the populator for inclusion into the active cache of published JSD entries. Each entry returned to the JSD is identified by an integer, known as an entry ID. The entry IDs are returned by the populator to the JSD. The JSD observes the integer as an opaque token unique to a single entry within the populator.

[0083] The first entry ID is used to kick-start the population process and is returned to the JSD by the getRootEntry() method. All subsequent entry IDs are returned by the following methods: getParentEntry(), getFirstChildEntry(), and getPeerEntry(). Each property associated with an entry is identified by a combination of an entry ID and a property name. Similarly, each attribute associated with an entry is identified by a combination of an entry ID and an attribute name. Each attribute associated with a property is identified by the topic of entry ID, property name, and attribute name.

[0084] Normally, serialization and compilation as described above are only able to translate a finite, pre-defined set of primitive Java™ Language data types. In one embodiment, then, the StaticTreeSerializer and StaticTreeCompiler classes can only read and write values in terms of the following primitive data types: string, boolean, int, long, short, byte, char, double, float, and arrays thereof. However, some object-oriented databases may define complex data types: that is, data types which combine one or more of the primitive data types. For example, a complex data type could be an object which includes a string, two long integers, and an array of boolean values. To modify the grammar to describe these complex data types would render the grammar large and unwieldy. Therefore, to enable serialization and compi-

lation to understand complex data types, in one embodiment a customization plug-in is provided.

[0085] In one embodiment, the plug-in comprises a Java™ Language interface called StaticTreePlugIn, which is defined as follows:

```

5      package javaos.javax.system.database.statictree;

      import java.io.*;
      import java.util.Hashtable;

10     public interface StaticTreePlugIn {

            public String compile(Hashtable properties) throws
15                IllegalArgumentException;
            public void serialize(Hashtable properties) throws
                IllegalArgumentException;
20     }

```

[0086] The StaticTreePlugIn interface should be implemented by a plug-in class for utilizing complex data types. In other words, developers who wish to utilize complex data types should design a particular plug-in class that implements the StaticTreePlugIn interface. In one embodiment, an example of such a plug-in is the "business card" discussed in conjunction with the database description grammar. The business card is a complex data type that includes a collection of properties and/or attributes for a particular software component.

[0087] In one embodiment, a plug-in functions as follows. When compiling a grammatical form that contains a complex data type, the compiler determines that the data type is not a known or recognized data type. The compiler then creates a hash table containing the primitive data types which make up the complex data type. In other words, the hash table contains the properties of the complex data type. The compiler then calls the compile() method of the plug-in and passes the hash table to the plug-in. The plug-in collapses the data types in the hash table down to a single, complex object. The compile() method of the plug-in returns the name of the complex property in String format. When serializing, on the other hand, the serializer passes a blank hash table to the serialize() method of the plug-in. The serialize() method of the plug-in should translate the complex object into a series of primitives and fill the hash table with the list of primitives.

[0088] Various embodiments of the present invention further include receiving or storing instructions and/or data implemented in accordance with the foregoing description upon a carrier medium. Suitable carrier mediums include storage mediums such as disk, as well as electrical signals or digital signals conveyed via a communication medium such as network 108 or a wireless link.

[0089] A computer program product for implementing the invention can be in the form of a computer program on a carrier medium. The carrier medium could be a storage medium, such as solid state magnetic optical, magneto-optical or other storage medium. The carrier medium could be a transmission medium such as broadcast, telephonic, computer network, wired, wireless, electrical, electromagnetic, optical or indeed any other transmission medium.

[0090] While the present invention has been described with reference to particular embodiments, it will be understood that the embodiments are illustrated and that the invention scope is not so limited. Any variations, modifications, additions and improvements to the embodiments described are possible. These variations, modifications, additions and improvements may fall within the scope of the invention.

Claims

1. A method for transforming transient contents of an object-oriented database into a persistent textual form according to a grammar, said object-oriented database and said textual form being stored in a memory of a computer system, said method comprising:

expressing a plurality of entries as objects in said object-oriented database in a transient form, wherein said entries and said objects relate to configuration parameters of software and hardware of said computer system; generating a textual form of said plurality of entries corresponding to said objects in said object-oriented data-

base by expressing said plurality of entries according to said grammar, wherein said textual form comprises a key state of said plurality of entries, wherein said textual form is lesser in size than said transient form of said plurality of entries in said object-oriented database;
storing said textual form in one or more persistent containers.

2. The method of claim 1,
wherein said objects of said object-oriented database pertain to one or more application programs.
3. The method of claim 1 or claim 2,
wherein said textual form describes a hierarchy of entries corresponding to a hierarchy of objects in said object-oriented database.
4. The method of any preceding claim,
wherein said grammar includes a set of keywords and a syntax.
5. The method of claim 4,
wherein said keywords are selected from the group consisting of: tree, entry, properties, and attributes.
6. The method of any preceding claim,
wherein said textual form includes name-value pairs corresponding to properties and attributes of said software and hardware of said computer system.
7. The method of any preceding claim,
wherein said generating a textual form includes locking said object-oriented database for exclusive access during said generating a textual form.
8. The method of any preceding claim,
wherein said generating a textual form further comprises flattening a hierarchy of objects into a byte stream, wherein said byte stream is smaller in size than said hierarchy of objects as stored in said object-oriented database in said memory of said computer system.
9. A computer program product comprising program instructions for transforming transient contents of an object-oriented database into a persistent textual form according to a grammar, wherein said program instructions are executable to implement:

expressing a plurality of entries as objects in said object-oriented database in a transient form, wherein said entries and said objects relate to configuration parameters of software and hardware of a computer system;
generating a textual form of said plurality of entries corresponding to said objects in said object-oriented database by expressing said plurality of entries according to said grammar, wherein said textual form comprises a key state of said plurality of entries, wherein said textual form is lesser in size than said transient form of said plurality of entries in said object-oriented database; and
storing said textual form in one or more persistent containers.
10. The computer program product of claim 9,
wherein said objects of said object-oriented database pertain to one or more application programs.
11. The computer program product of claim 9 or claim 10,
wherein said textual form describes a hierarchy of entries corresponding to a hierarchy of objects in said object-oriented database.
12. The computer program product of any one of claims 9 to 11,
wherein said grammar includes a set of keywords and a syntax.
13. The computer program product of claim 12,
wherein said keywords are selected from the group consisting of: tree, entry, properties, and attributes.
14. The computer program product of any one of claims 9 to 13,
wherein said textual form includes name-value pairs corresponding to properties and attributes of said software

and hardware of said computer system.

15. The computer program product of any one of claims 9 to 14,
wherein in said generating a textual form, said program instructions are further executable to lock said object-oriented database for exclusive access during said generating a textual form.
16. The computer program product of any one of claims 9 to 15,
wherein in said generating a textual form, said program instructions are further executable to flatten a hierarchy of objects into a byte stream, wherein said byte stream is smaller in size than said hierarchy of objects as stored in said object-oriented database in said memory of said computer system.
17. The computer program product of any one of claims 9 to 16, embodied on a carrier medium.
18. The computer program product of claim 17, wherein the carrier medium is a storage medium.
19. The computer program product of claim 17, wherein the carrier medium is a transmission medium.
20. A computer system for transforming transient contents of an object-oriented database into a persistent textual form according to a grammar, said computer system comprising:
 - a CPU;
 - a memory coupled to said CPU;
 - wherein said memory stores said object-oriented database;
 - wherein said memory stores program instructions executable by said CPU, wherein said program instructions are executable to:
 - express a plurality of entries as objects in said object-oriented database in a transient form, wherein said entries and said objects relate to configuration parameters of software and hardware of a computer,
 - generate a textual form of said plurality of entries corresponding to said objects in said object-oriented database by expressing said plurality of entries according to said grammar, wherein said textual form comprises a key state of said plurality of entries, wherein said textual form is lesser in size than said transient form of said plurality of entries in said object-oriented database; and
 - store said textual form in one or more persistent containers.
21. The computer system of claim 20,
wherein said objects of said object-oriented database pertain to one or more application programs.
22. The computer system of claim 20 or claim 21,
wherein said textual form describes a hierarchy of entries corresponding to a hierarchy of objects in said object-oriented database.
23. The computer system of any one of claims 20 to 22,
wherein said grammar includes a set of keywords and a syntax.
24. The computer system of any one of claims 20 to 23,
wherein said textual form includes name-value pairs corresponding to properties and attributes of said software and hardware of said computer system.
25. The computer system of any one of claims 20 to 24,
wherein in said generating a textual form, said program instructions are further executable to lock said object-oriented database for exclusive access during said generating a textual form.
26. The computer system of any one of claims 20 to 25,
wherein in said generating a textual form, said program instructions are further executable to flatten a hierarchy of objects into a byte stream, wherein said byte stream is smaller in size than said hierarchy of objects as stored in said object-oriented database in said memory.

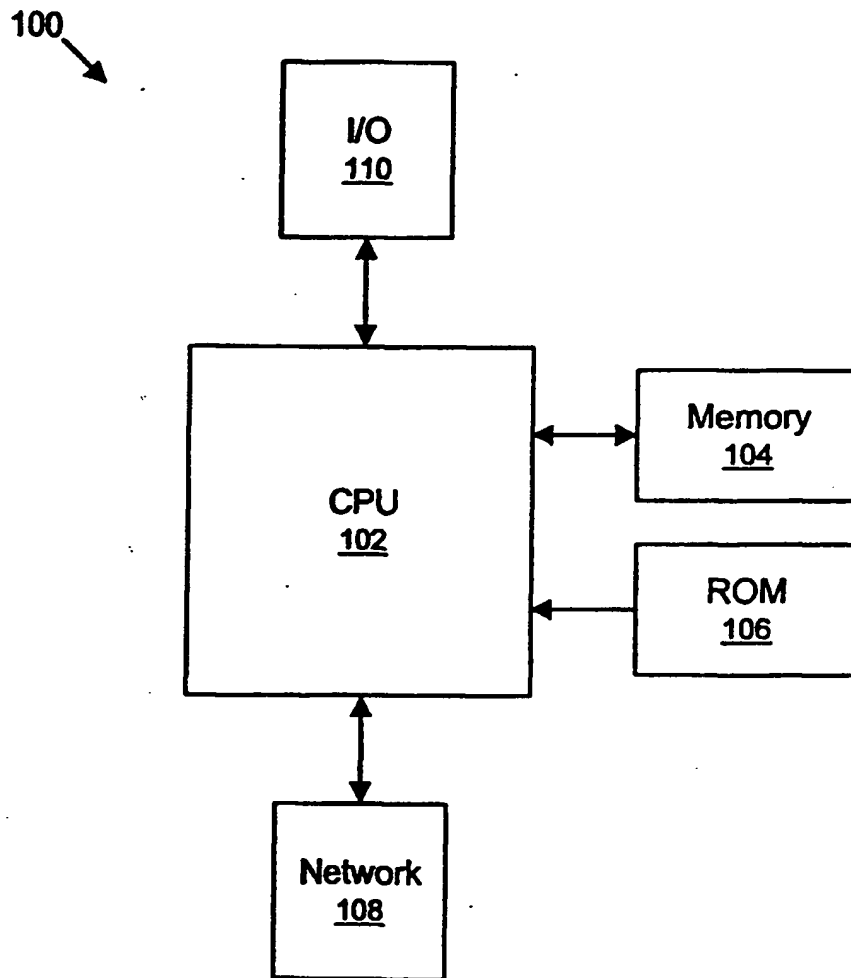


FIG. 1 (PRIOR ART)

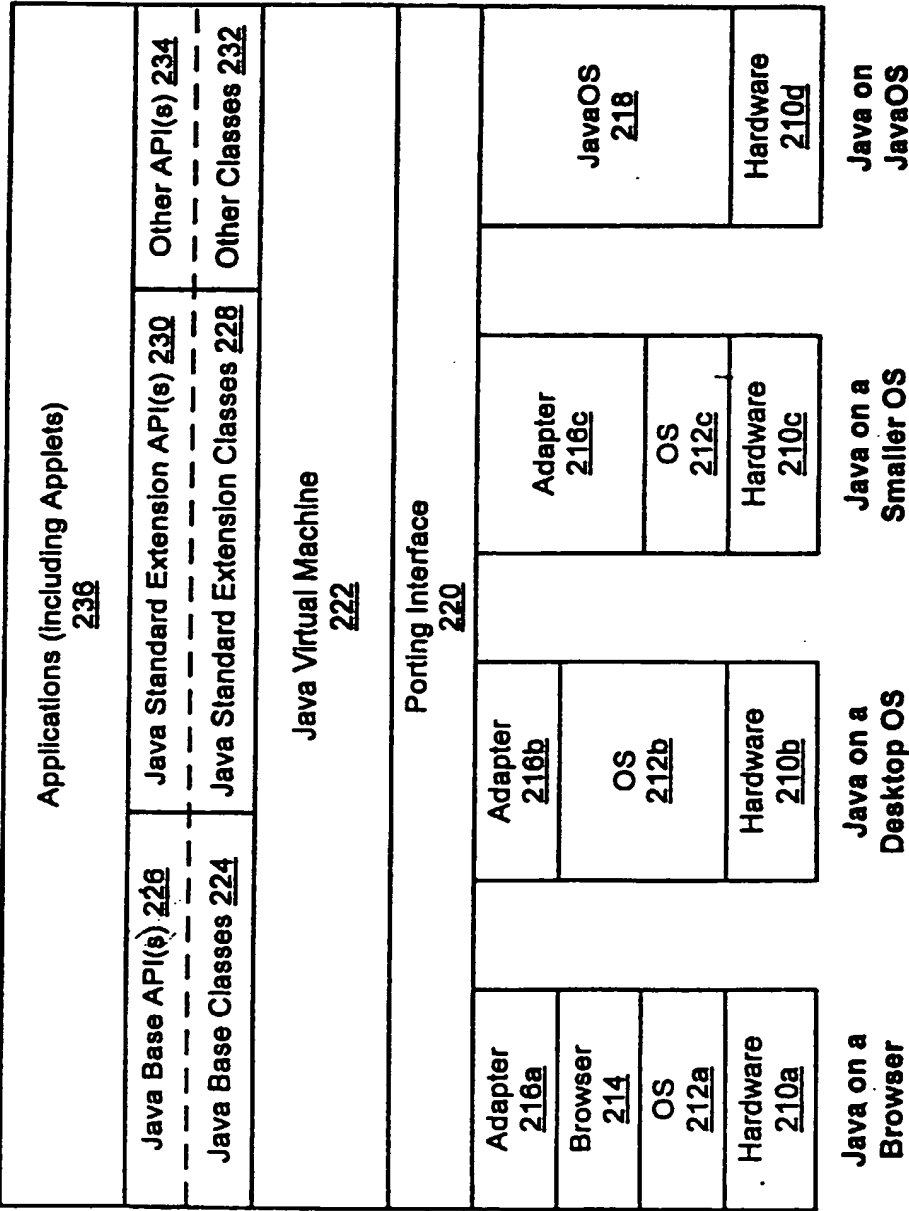


FIG. 2 (PRIOR ART)

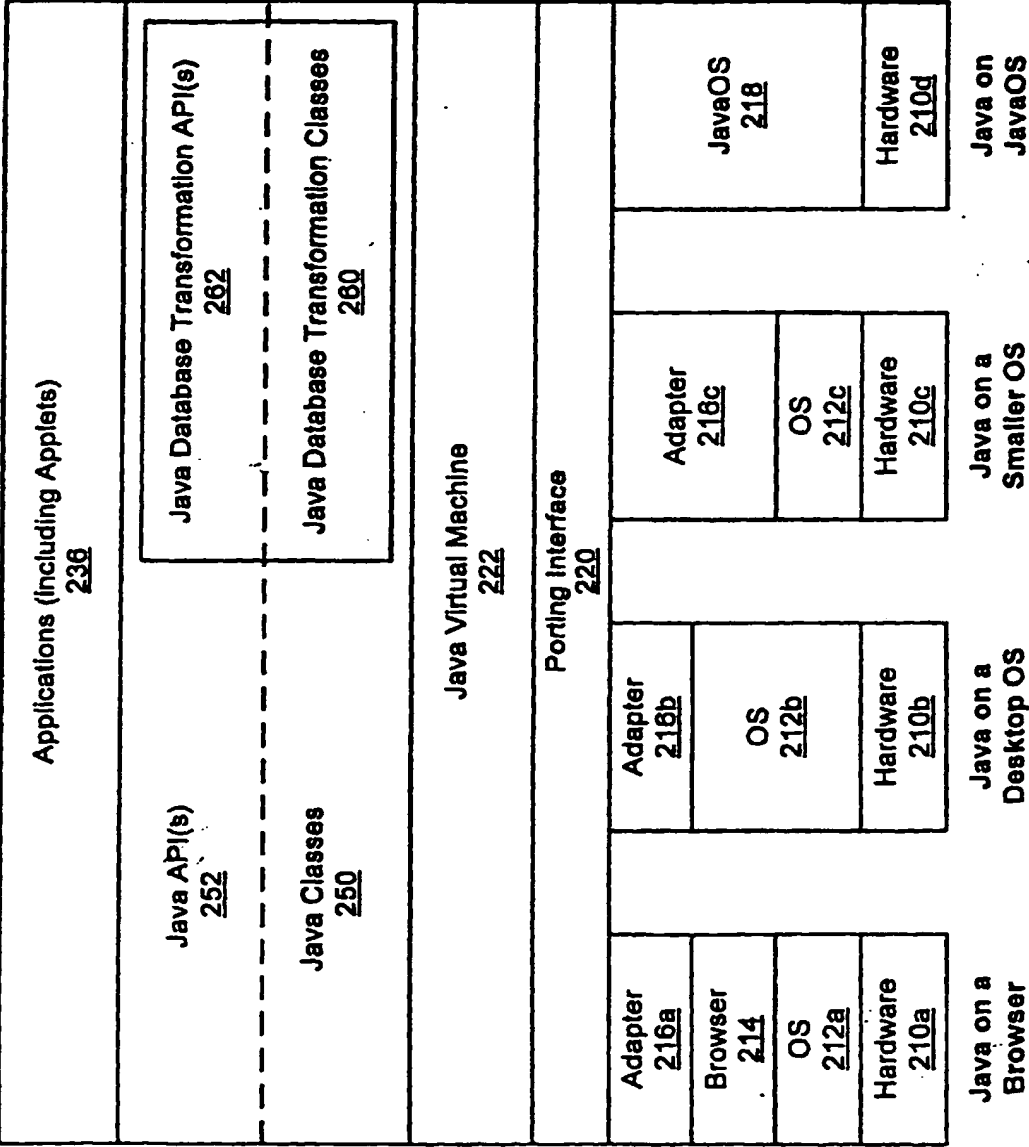
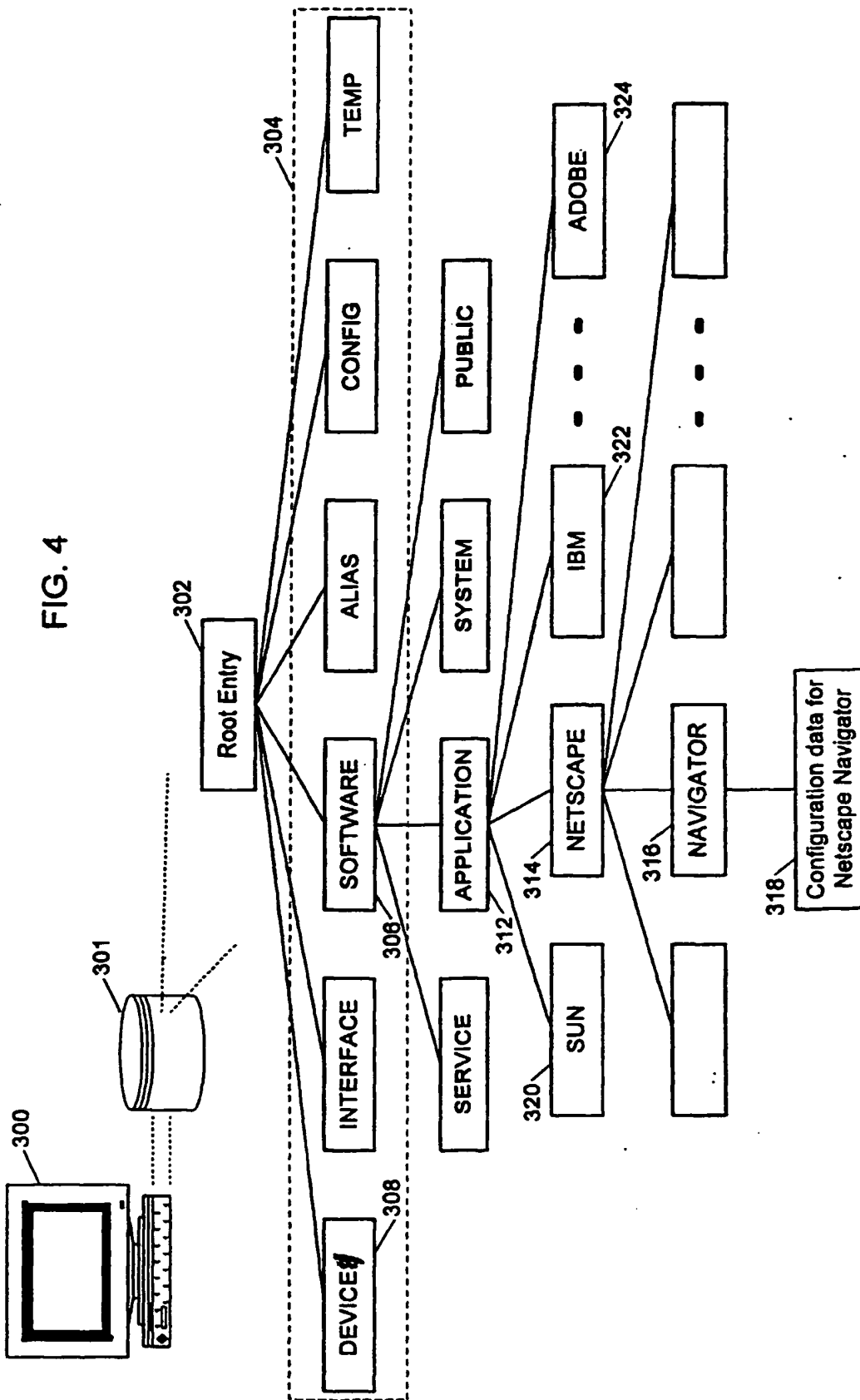


FIG. 3



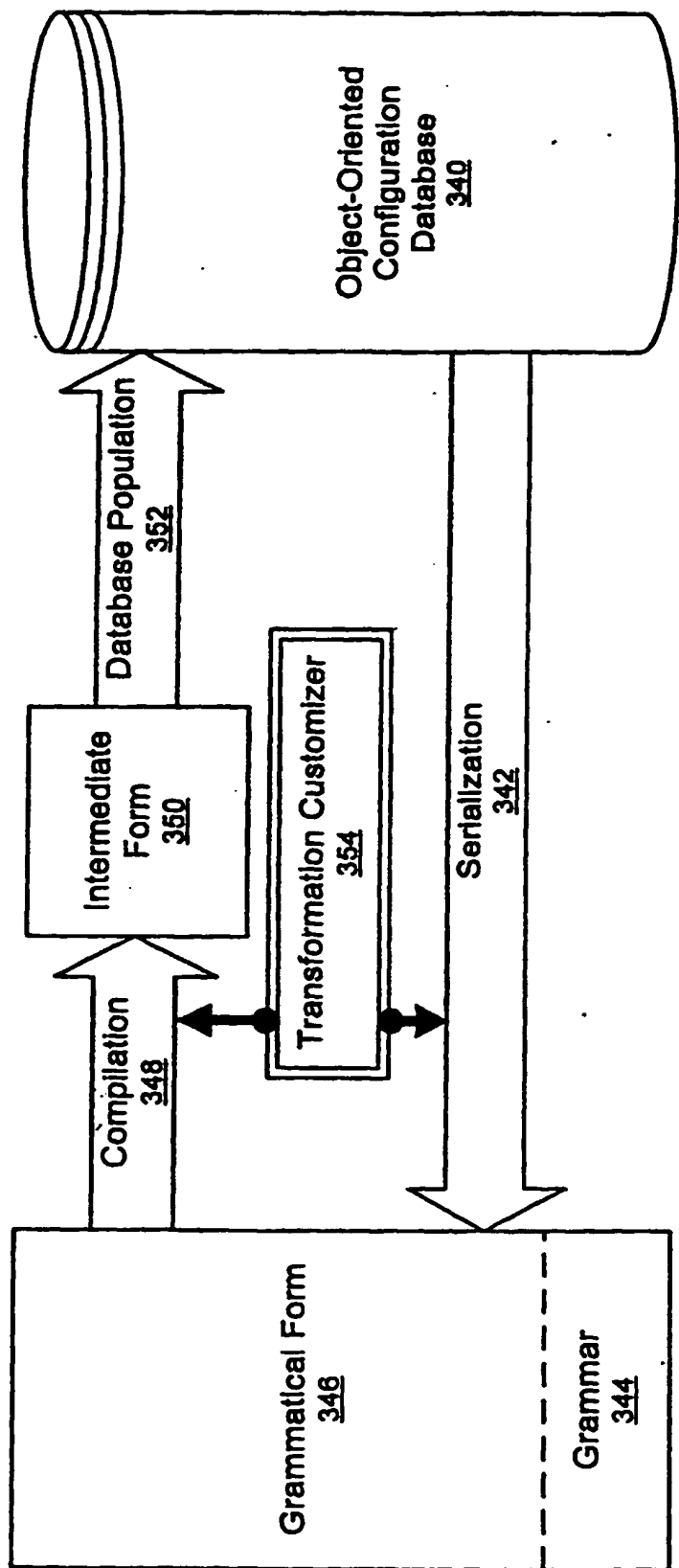


FIG. 5

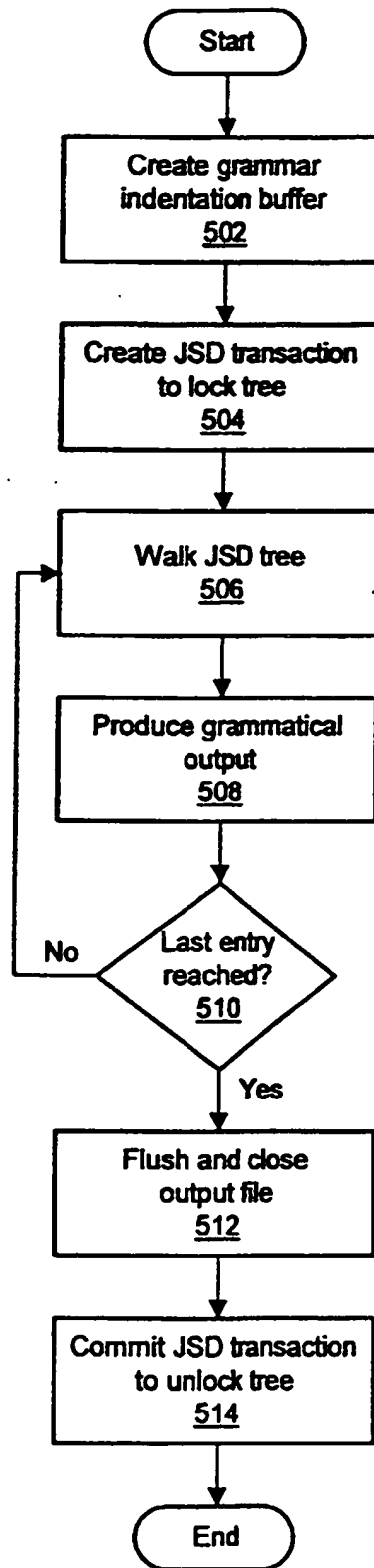


FIG. 6

FIG. 7

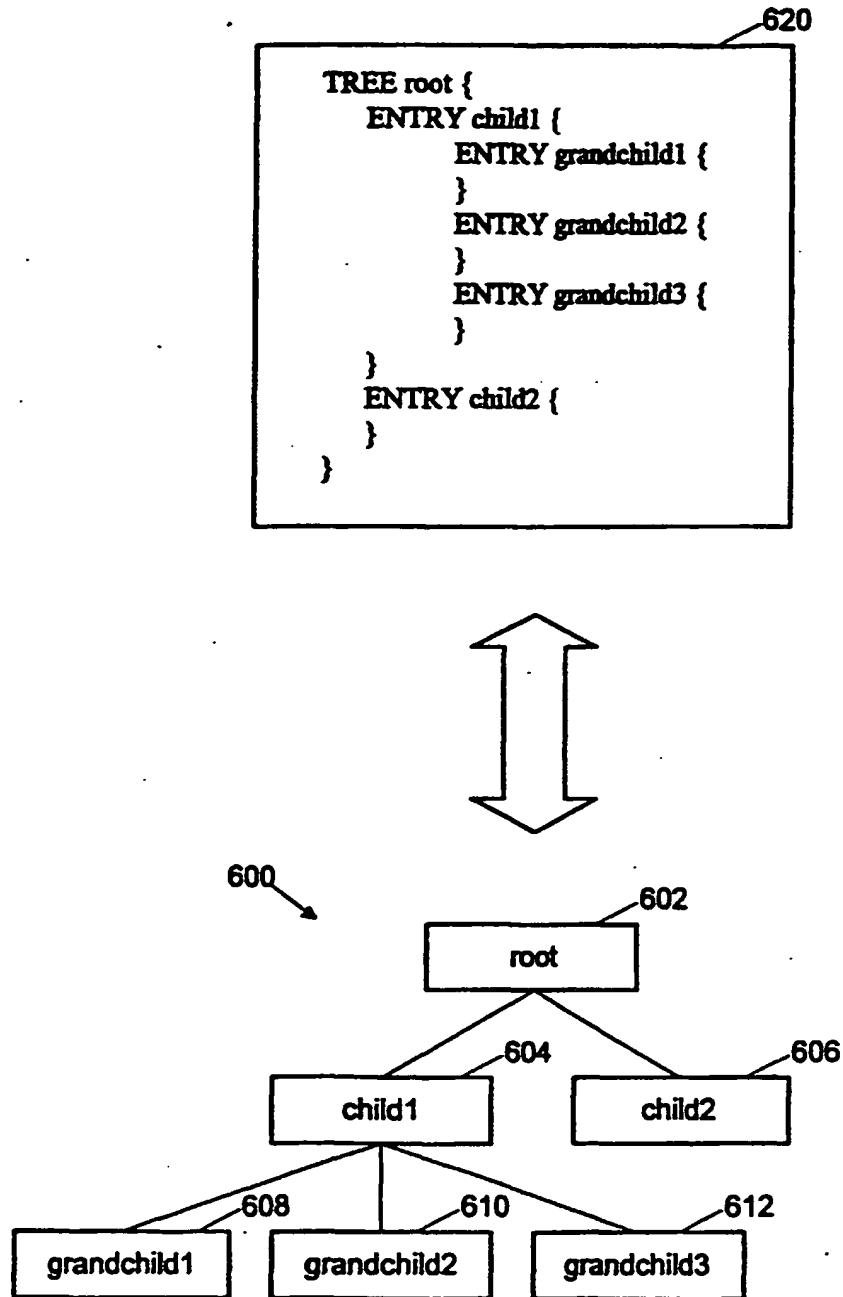


FIG. 8

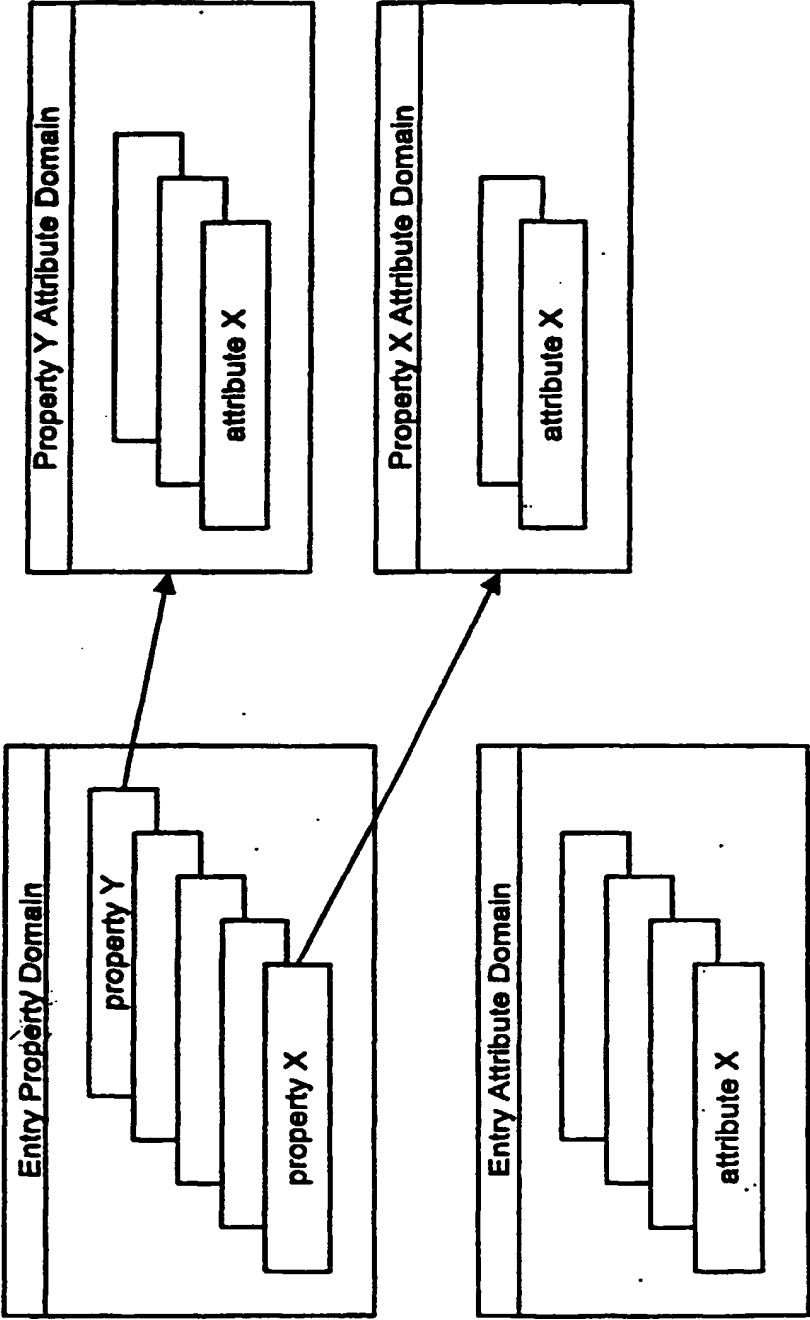
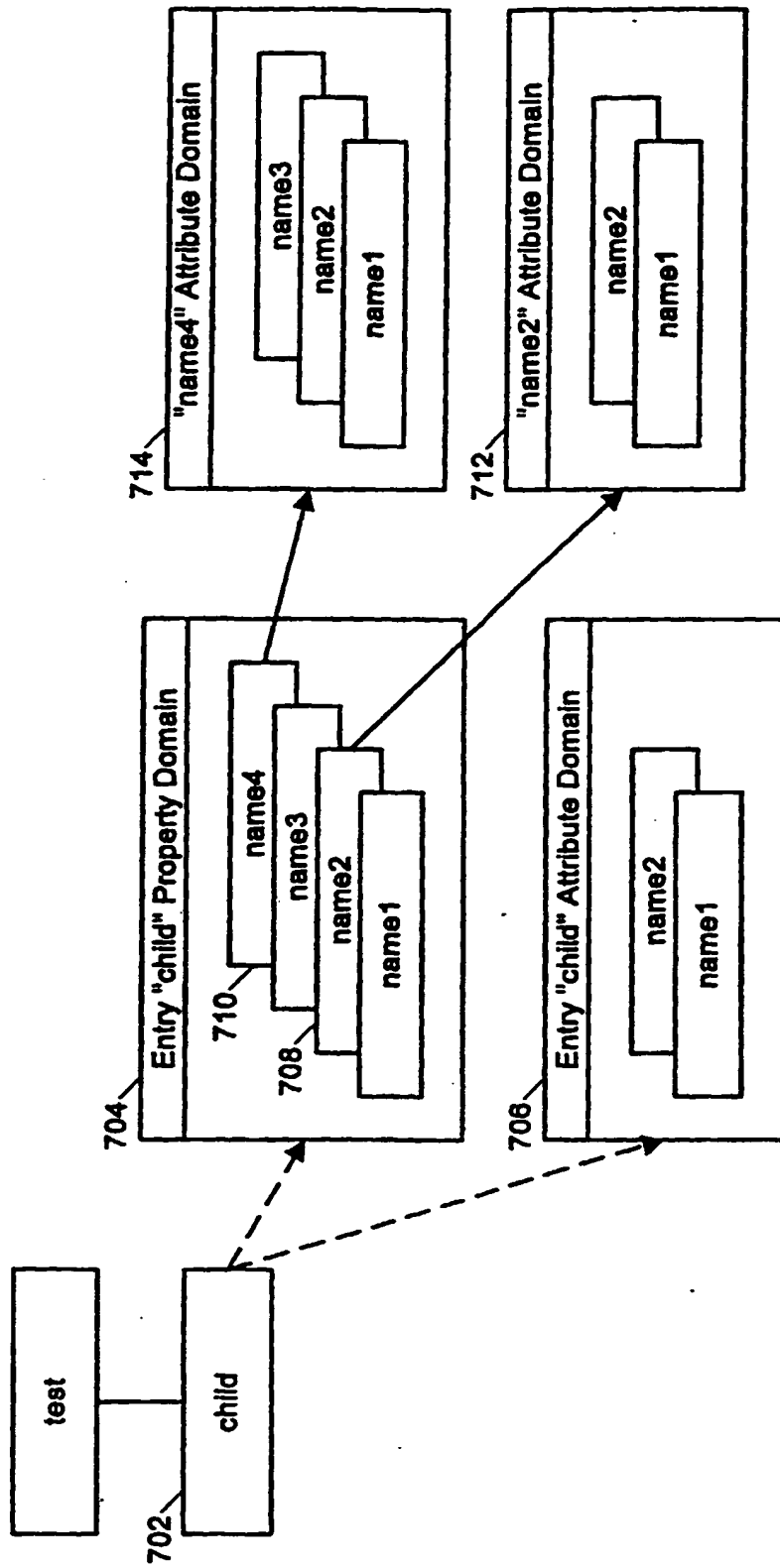


FIG. 9



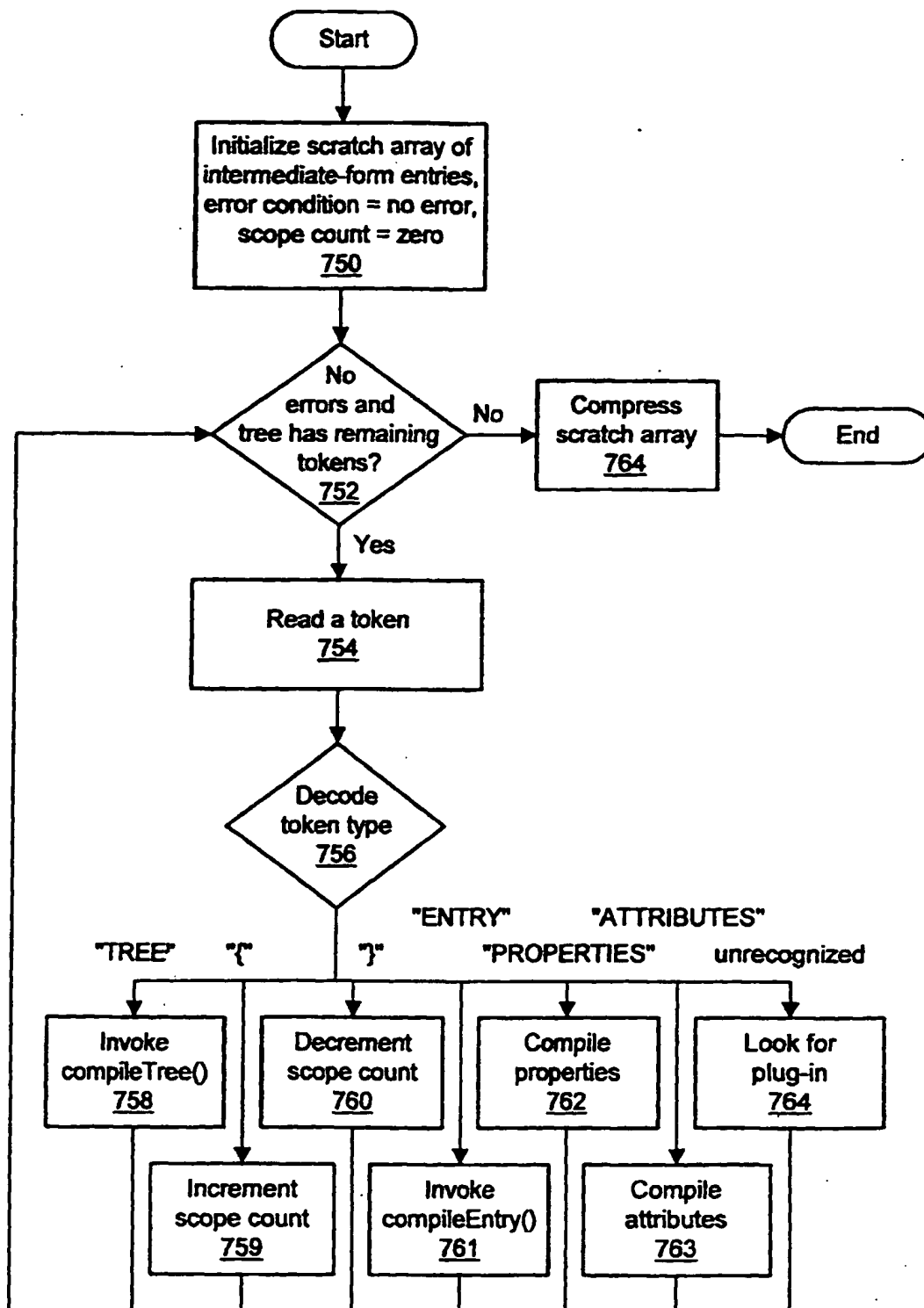


FIG. 10

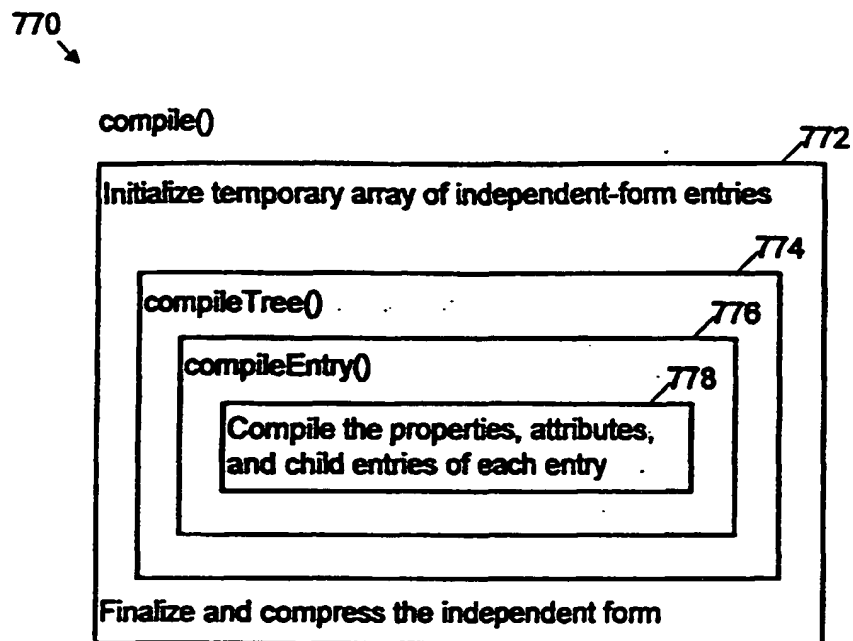


FIG. 11



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 00 30 1149

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	US 5 758 154 A (QURESHI IMRAN I) 26 May 1998 (1998-05-26) * the whole document *	1-3, 9-11, 20-22	G06F17/30
A	US 5 694 598 A (DURAND JACQUES ET AL) 2 December 1997 (1997-12-02) * abstract * * figures 2-4 * * column 2, line 42 - column 3, line 39 *	1,9,20	
A	EP 0 631 229 A (IBM) 28 December 1994 (1994-12-28) * page 4, line 32 - page 6, line 42 *	1,9,20	
A	US 5 414 812 A (LEE LUCILLE C ET AL) 9 May 1995 (1995-05-09) * abstract *	1,9,20	
A	MYERS B A ET AL: "ENVIRONMENT FOR RAPIDLY CREATING INTERACTIVE DESIGN TOOLS" VISUAL COMPUTER, DE, SPRINGER, BERLIN, vol. 8, no. 2, 1 February 1992 (1992-02-01), pages 94-116, XP000374123 ISSN: 0178-2789 * page 102, line 10 - page 103, line 20 *	1,9,20	<div>TECHNICAL FIELDS SEARCHED (Int.Cl.7)</div> <div>G06F</div>
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 14 June 2000	Examiner Abbing, R
<div>CATEGORY OF CITED DOCUMENTS</div> <div> X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document </div>			

EPO FORM 1503 03/82 (P04/C01)

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 00 30 1149

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

14-06-2000

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5758154	A	26-05-1998	NONE		
US 5694598	A	02-12-1997	NONE		
EP 0631229	A	28-12-1994	US	5797007 A	18-08-1998
			JP	2711220 B	10-02-1998
			JP	7098649 A	11-04-1995
US 5414812	A	09-05-1995	NONE		

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

Set	Items	Description
S1	1298184	CONFIGURAT??? OR AUTOCONFIGURAT??? OR TOPOLOGY OR TOPOLOGI- ES OR LAYOUT? ?
S2	23239134	COMPUTER? ? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?
S3	22359561	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDE- NTIFI??????
S4	9094945	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BAN- K? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABAN- K? ? OR TABLE? ? OR FILE? ?
S5	2422756	(SECOND??? OR OTHER OR ANOTHER OR ADDITIONAL OR EXTRA OR T- WO OR 2ND) (3N) (GROUP? ? OR TYPE? ? OR RANGE? ? OR ARRAY OR CO- LLECTION OR CLASSIFIC???? OR DESCRIPTOR? ? OR CATEGOR??? OR SUBTYPE? ? OR MODEL? ? OR CLASS? ?)
S6	315	((SEARCH??? OR QUERY??? OR QUERI?? OR SCAN OR SCANNING OR - SCANNED OR LOOKUP OR LOOK()UP OR RETRIEV??? OR LOCAT??? OR M- ATCH???) (3N) (S1 OR S2) (3N) S3 (3N) S4 (3N) S5) AND (PD<20020407 OR PY<2003)
S7	11	S6 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR T- ECHNOLOGY OR HARDWARE? ?) AND S4)/TI
S8	78784	((SEARCH??? OR QUERY??? OR QUERI?? OR SCAN OR SCANNING OR - SCANNED OR LOOKUP OR LOOK()UP OR RETRIEV??? OR LOCAT??? OR M- ATCH???) (100N) (S1 OR S2) (100N) S3 (100N) S4 (100N) S5) AND (PD<200- 20407 OR PY<2003)
S9	0	(S8 AND ((CONFIGURATION()DATABASE? ?)/TI)) NOT S7
S10	10	(CONFIGURATION()DATABASE? ?)/TI
S11	426	(S8 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR - TECHNOLOGY OR HARDWARE? ?) (3N) S4)/TI) NOT (S7 OR S10)

? show files

File 275:Gale Group Computer DB(TM) 1983-2006/Feb 03
(c) 2006 The Gale Group

File 47:Gale Group Magazine DB(TM) 1959-2006/Feb 03
(c) 2006 The Gale group

File 16:Gale Group PROMT(R) 1990-2006/Feb 03
(c) 2006 The Gale Group

File 624:McGraw-Hill Publications 1985-2006/Feb 06
(c) 2006 McGraw-Hill Co. Inc

File 484:Periodical Abs Plustext 1986-2006/Feb W1
(c) 2006 ProQuest

File 613:PR Newswire 1999-2006/Feb 06
(c) 2006 PR Newswire Association Inc

File 813:PR Newswire 1987-1999/Apr 30
(c) 1999 PR Newswire Association Inc

File 239:Mathsci 1940-2006/Mar
(c) 2006 American Mathematical Society

File 370:Science 1996-1999/Jul W3
(c) 1999 AAAS

File 696:DIALOG Telecom. Newsletters 1995-2006/Feb 06
(c) 2006 Dialog

File 621:Gale Group New Prod. Annou. (R) 1985-2006/Feb 03
(c) 2006 The Gale Group

File 674:Computer News Fulltext 1989-2005/Oct W2
(c) 2005 IDG Communications

File 68:Gale Group Business A.R.T.S. 1976-2006/Jan 31
(c) 2006 The Gale Group

File 369:New Scientist 1994-2006/Aug W4
(c) 2006 Reed Business Information Ltd.

File 160:Gale Group PROMT(R) 1972-1989
(c) 1999 The Gale Group

File 635:Business Dateline(R) 1985-2006/Feb 04
(c) 2006 ProQuest Info&Learning

FULL TEXT
NPL

File 15:ABI/Inform(R) 1971-2006/Feb 06
(c) 2006 ProQuest Info&Learning
File 9:Business & Industry(R) Jul/1994-2006/Feb 03
(c) 2006 The Gale Group
File 13:BAMP 2006/Jan W5
(c) 2006 The Gale Group
File 810:Business Wire 1986-1999/Feb 28
(c) 1999 Business Wire
File 610:Business Wire 1999-2006/Feb 06
(c) 2006 Business Wire.
File 647:CMP Computer Fulltext 1988-2006/Feb W3
(c) 2006 CMP Media, LLC
File 98:General Sci Abs/Full-Text 1984-2004/Dec
(c) 2005 The HW Wilson Co..
File 148:Gale Group Trade & Industry DB 1976-2006/Feb 03
(c) 2006 The Gale Group
File 634:San Jose Mercury Jun 1985-2006/Feb 04
(c) 2006 San Jose Mercury News
File 256:TECINFOSOURCE 82-2005/DEC
(c) 2006 INFO.SOURCES INC

11/9/118 (Item 118 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2006 The Gale Group. All rts. reserv.

01358981 SUPPLIER NUMBER: 08238632 (THIS IS THE FULL TEXT)
An end to cable chaos; a system that fully integrates CAD and relational data base technology provides the first proactive system for communications management on a large scale.

Bloom, Gregory
Telecommunications, v24, n2, p54(5)
Feb, 1990

ISSN: 0278-4831 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 2719 LINE COUNT: 00230

ABSTRACT: Cable chaos, resulting from inefficient or nonexistent tracking of a network's history, can major financial and operational impacts. A comprehensive communications management system linking all communications resources including cable can shorten project turnaround times, provide management control for improved decision making and reduce the cost of changes. Such a system integrates computer-aided design's (CAD) graphics capabilities with a relational data base management system, presenting information in both graphic and textual forms. Comprehensive systems offer management capabilities including creation and automated numbering of entities, data base query, patching, circuit reconfiguration, automatic work order creation, automatic schematic creation, and reporting facilities.

TEXT:

An End to Cable Chaos

Want to wrestle daily with a writhing snake that's 1000 miles long? Not many people would take that job, but that's the magnitude of today's challenge in cable and communications management. The term cable chaos describes the visible result of the incessant moves, adds, and changes that occur within dynamic voice and data networks .

A new generation of information technology promises to tame the beast by providing a comprehensive design and documentation system for control and management of all communications resources. This type of system integrates the proven graphics capabilities of CAD with the power and flexibility of relational data base management system (DBMS) technology for a complete picture of a communications environment.

With such a comprehensive system, it is now possible to collect, manage, and control all the information relating to cabling, connections, and telecommunications assets. Even the largest multinational communications operations can be brought under control.

THE PROBLEM: LITTLE CABLING DATA

The problem of cable chaos comes down to inefficient or nonexistent tracking of a network's history and growth. Often, so little is known about existing premises wiring that new cable is installed as a matter of routine. It's no wonder that risers and cable trays are filled to capacity -- even in office buildings that were only recently constructed. Some companies have so despaired of dealing with the cable chaos in existing buildings that moving the entire operation to another site and starting over can seem to be the least expensive alternative!

Cable chaos causes major financial and operational impacts. Inefficient management of communications assets can add up to many millions of dollars annually, and -- in the most serious scenario -- can bring business operations to a standstill. The consequences and costs include:

- * direct network maintenance and changed management costs
- * poor asset management

* risk of business interruption.

Each move, add, or change in an undocumented network can cost between \$1000 and \$1500. Unfortunately, the communications manager can be certain that some portion of this expense is unnecessary. But without adequate information, there is no basis for finding and eliminating the waste. Expenses are sure to be high because project turnaround times are long and there is no management control.

From a wider facilities perspective, needless redundancy of cabling means wasted cableway space. Also, since there is probably a lack of information on the installed equipment base, utilization of all related communications assets will be inefficient. The problem can get so bad that ceilings literally fall from the weight of unused cabling.

SOLUTION: COMPREHENSIVE MANAGEMENT

The goal of a truly comprehensive communications management system (see Figure 1) is to link all communications resources -- not just cable -- to all corporate facilities for improved and complete asset management. The results can be reducing the cost of changes, shortening project turnover times, and providing management control for improved decision making.

Integration of telecommunications information within an organization-wide facilities management (FM) data base is needed because:

- * capital assets for telecommunications represent a significant portion of corporate investment in facilities
- * information on facilities, including floor plans, is crucial to communications planning and management
- * moves, adds, and changes in a network are often motivated by changes in facility utilization
- * an integrated (FM) data base allows efficient network administration across multiple sites.

A comprehensive communications management system can be achieved by capturing and representing all communications resources in a single, integrated data base. These comprehensive systems can be used to coordinate efficiently the processes of network planning, design, configuration, and change management. The move towards such integrated systems is now widespread. An example is Command (Communications Management and Design) from Isicad, which serves as a model for the following discussion.

INTEGRATION OF GRAPHICS AND TEXT

The comprehensive system's integrated data base contains information on communications resources as both graphic and alphanumeric (text) data.

A common area of understanding between telecommunications and FM functions is typically the floor plan (see Figure 2). Therefore, a CAD-based graphic system is a natural interface between facilities and telecommunications information. Network information shown graphically on floor plans includes devices and equipment, outlets, cable, and cableways.

RELATIONAL DBMS

In communications management systems, the advantages of a relational DBMS include linkage of graphics and text, data exchange, and flexible reporting. Integration of graphics and text saves work and enhances user understanding of the system and the network it represents. Changes to CAD drawings, such as floor plans, automatically update corresponding alphanumeric attributes. For example, adding the symbol of a computer terminal updates the corresponding text description and equipment listings. Conversely, changing an entry in an alphanumeric listing changes the affected symbols or cable routing on the drawings.

For compatibility among different computer environment and for portability of information, it is advantageous if the underlying relational DBMS in the system is a standard, commercially available package rather than a proprietary module. For ease of integration with existing FM and architectural CAD data bases, it is also important for the system to support graphic data exchange in different CAD data formats, such as IGES, SIF, and DXF. Thus, existing CAD drawings can be imported, and modified drawings can be exported. Benefits include savings in data entry time, as

well as enhanced information sharing with other FM functions.

Relational DBMS technology has exceptionally flexible reporting capabilities. Attributes residing in the system can be queried, analyzed, and reported in a variety of ways. These reporting functions save time and labor by automating the paperwork requirements of routine communications management. The system can generate all kinds of documentation, including schematic diagrams, equipment and cable schedules, bills of materials, and work orders.

Reporting capabilities are further enhanced through built-in links to business graphics and spreadsheet modules (see Figure 3) -- an additional benefit of using an off-the-shelf data base. Use of business graphics and spreadsheet modules provides valuable tools for management decision making. For example, it is possible both to track historic resource utilization and to analyze that information as a basis for future planning. With these decision support tools, communications management can become truly proactive -- looking ahead of the usage curve -- rather than merely attempting to keep up with changes.

THE SYSTEM AT WORK

Specific communications management capabilities of comprehensive systems include:

- * creation and automated numbering of entities
- * data base query
- * patching
- * circuit reconfiguration
- * automatic work order creation
- * automatic schematic creation
- * reporting.

Creating and Automated Numbering of Entities

Working with the system begins with designing a network or documenting an existing one. Here, the extraordinary usefulness of the CAD capabilities becomes immediately obvious. Communication devices, cableways, cables, cable routes, and connections may be identified or defined, placed, and automatically numbered on CAD drawings. In the same operation, corresponding attributes are registered in the data base automatically.

CAD drawings can be either imported from other CAD systems, as described previously, or drawn within the communications management system when designing a network. To speed the process, equipment units are added to CAD drawings as symbols. The system's symbol library includes such items as specific PC models and telecom outlets. Symbols also may be defined by the user so that corporate standards can be followed. Each symbol has a set of attributes, including specifications such as manufacturer, model number, and cost which are added to the data base.

With these customized tools, the design phase moves quickly. Experience designing a network for 10-story bank facility has shown that a job typically requiring 24 man-years for the design phase could be reduced to 0.5 man-year with this CAD-based system.

Numbering entities is critical for tracking resources. Integrated systems assign numbers automatically to new entities and can adhere to the user's predefined format for compatibility with existing identification schemes. Again, the process of adding the symbols to the drawing also registers their identification numbers and attributes into the data base. Numbers then become important key field identifiers for subsequent searching within the DBMS.

As an example, consider the addition of two new PCs. The system prompts for a device number as a symbol for each unit that is placed in the drawing. If 20 terminals are already in the data base and two are being added, the system assigns the numbers 21 and 22. The corresponding telecom outlet symbols are added and numbered in similar fashion.

Keeping track of cable paths is fundamental to cable management decisions, and yet few systems until now have been able to perform this function. Through comprehensive systems, cable paths can be created and

viewed graphically on the floor plan, as schematics or as listings of routes and connections. To assist the designer in layout decisions, the system can flag design rule violations, such as when a cableway capacity is being exceeded as cable is routed. Further, the user can analyze the effects of disrupting a specific cableway, whether the disruption is intentional (as in making changes) or accidental (as in troubleshooting and disaster recovery).

In a single operation, cables can be laid out individually or, for example, as blocks of 50 twisted pairs. Either way, the system will assign and register an identifier for each cable. Cables may be of any type, including user-defined types, and may even be logical entities, such as microwave links.

Entities such as devices and cables can be added to drawing in any order. Establishing connections is then a straightforward procedure. For cable assignment, the system prompts with FROM and searches patch-panel entries in the data base for the first open socket. This selection is presented to the user, who can accept it or override it. The next prompt, TO, asks for the terminal device, which is selected simply by either pointing to the symbol with the graphics cursor or by entering the alphanumeric identifier. As selections are made, the system builds a cable routing schedule. This schedule contains total cable length, cable type, and other relevant data.

Data Base Query

As cable layout takes shape during the network design phase, the user is building an ongoing data base. This data base provides connectivity information that is essential both to completing the design phase and to managing subsequent changes (see Table 1).

Through the relational DBMS, the user can request information in a wide variety of ways from the data base. In the system used as a model, the nontechnical question-and-answer style of the DBMS query language is similar to IBM SQL, with which many users are already comfortable.

To further promote ease of use, the system's graphic capabilities make it possible for the user merely to point to objects on a drawing display to obtain attribute listings. The ability to view entities either graphically or as text increases understanding and helps assure that updates are kept current.

An example of interrogating the data base might be to obtain an equipment schedule, perhaps for asset evaluation. This schedule would show all devices and outlets within the system and their attributes, including cost. Also included can be an item for tracking online status, such as:

STATUS -- ORDERED, NOT INSTALLED

Other pertinent data on this listing might include the department, so that any moves, adds, and changes can be charged back to the appropriate cost center.

The user might also wish to determine whether a specific cableway is overloaded. For any cable in the network, the system can report its type, category, description, diameter, weight, and so on. Based on cable diameter, the system can calculate whether the cross-sectional area of a cableway is nearing capacity.

Data base query is especially valuable for trouble-shooting and disaster recovery. If a cable is damaged, the system can report instantly -- with flashing symbols on the graphic display -- all the affected resources, right down to specific sockets on patch panels and terminal outlets. FM resources that are critical to disaster recovery can also be tied to the data base, including fire, security, and environmental control systems.

Patching

Much of the effort in conventional cable management involves identifying and tracking patch-panel connections throughout the network. As shown in Figure 4, comprehensive systems can assign cables to specific

ports on patch panels. Port assignments on each panel can be viewed and/or changed at an administrator's terminal or at a graphic workstation. The system's implementation of patching is closely related to actual patching practice, making it easy and self-explanatory.

Circuit Reconfiguration

Moves, adds, and changes occupy much of the attention of communications managers. This work is costly, time-intensive, and error-prone. Through comprehensive communications management systems, proposed circuit changes can be made by disconnecting and reconnecting cables among devices on the drawings. Making changes on the drawing, in turn, automatically generates a schematic as well as a work order. To simplify the process and to prevent errors, menu-driven routines prompt the user through each step in responding to a request for a move, add, or change.

As moves, adds, and changes are entered into the **data base**, the system handles all aspects of updating and reconfiguring the network representation so that documentation always reflects the current status. A change log is maintained so that there is an orderly record of all transactions affecting the network.

Automatic Work Order Creation

Generation of work orders automatically by the system assures that reconfiguration requirements are communicated accurately to the employee who will actually do the physical work. Work orders can be **tracked** through their life cycles--from proposed, to approved, to implemented--thereby providing managers with an accurate picture of work in progress.

Work order document form an audit trail describing changes to the network over time. This documentation also serves as an aid to troubleshooting, a complete maintenance history, and a basis for network analysis.

Automatic Schematic Creation

A graphic schematic is an effective means of visualizing the effect of reconfigurations. The system can create graphic schematics "on the fly," or as reconfiguration functions are being performed.

Reporting

Data base query, as already discussed, can be a freeform, question-and-answer process. Some comprehensive systems also support more formal reporting, including both standardized, predefined reports and user-defined reports. The objective is to assist the user in making informed decisions, presenting findings to others, and documenting all aspects of communications management.

Standard reports can be selected from menus so that the infrequent user or manager can easily find items of interest. In select systems, even customized reporting capabilities exist.

DRAWING-LEVEL VS. PROJECT-LEVEL

Some currently available cable management systems amount to specialized enhancements of architectural CAD tools. There is not the capability to manage other types of resources, nor may there be any integration with the larger FM environment. Such systems might be termed drawing-level solutions, since the management perspective stops at the boundaries of an isolated floor plan.

Integration of floors, sites, and campuses can be achieved through the comprehensive systems described here, since they integrate views of both local and remote resources. For example, when disconnecting a terminal, the corresponding host and mux can be **identified** -- whether **located** on another floor, in another building, or on a different campus halfway across the world. Since the system tracks logical as well as physical links, microwave channels can also be traced. Tracing all affected resources permits their prompt reassignment, thereby promoting efficient utilization of assets. Such a comprehensive communications management

systems can truly be called a project-level solution for dealing with cable chaos.

Gregory Bloom is the vice president of Isicad, Inc.

CAPTIONS: Comprehensive communications management system. (chart); Implementation costs. (graph); Design cable schedule. (table)

COPYRIGHT 1990 Horizon House Publications Inc.

SPECIAL FEATURES: illustration; chart; graph; table

DESCRIPTORS: Relational Data Base Management Systems; Cables; Communications Management; Computer-Aided Design

FILE SEGMENT: TI File 148

?

7/9/10 (Item 2 from file: 148)
DIALOG(R) File 148:Gale Group Trade & Industry DB
(c)2006 The Gale Group. All rts. reserv.

04587736 SUPPLIER NUMBER: 08238632 (THIS IS THE FULL TEXT)

An end to cable chaos; a system that fully integrates CAD and relational data base technology provides the first proactive system for communications management on a large scale.

Bloom, Gregory

Telecommunications, v24, n2, p54(5)

Feb, 1990

ISSN: 0278-4831

LANGUAGE: ENGLISH

RECORD TYPE: FULLTEXT; ABSTRACT

WORD COUNT: 2719

LINE COUNT: 00230

ABSTRACT: Cable chaos, resulting from inefficient or nonexistent tracking of a network's history, can major financial and operational impacts. A comprehensive communications management system linking all communications resources including cable can shorten project turnaround times, provide management control for improved decision making and reduce the cost of changes. Such a system integrates computer-aided design's (CAD) graphics capabilities with a relational data base management system, presenting information in both graphic and textual forms. Comprehensive systems offer management capabilities including creation and automated numbering of entities, data base query, patching, circuit reconfiguration, automatic work order creation, automatic schematic creation, and reporting facilities.

TEXT:

An End to Cable Chaos

Want to wrestle daily with a writhing snake that's 1000 miles long? Not many people would take that job, but that's the magnitude of today's challenge in cable and communications management. The term cable chaos describes the visible result of the incessant moves, adds, and changes that occur within dynamic voice and data networks.

A new generation of information technology promises to tame the beast by providing a comprehensive design and documentation system for control and management of all communications resources. This type of system integrates the proven graphics capabilities of CAD with the power and flexibility of relational data base management system (DBMS) technology for a complete picture of a communications environment.

With such a comprehensive system, it is now possible to collect, manage, and control all the information relating to cabling, connections, and telecommunications assets. Even the largest multinational communications operations can be brought under control.

THE PROBLEM: LITTLE CABLING DATA

The problem of cable chaos comes down to inefficient or nonexistent tracking of a network's history and growth. Often, so little is known about existing premises wiring that new cable is installed as a matter of routine. It's no wonder that risers and cable trays are filled to capacity -- even in office buildings that were only recently constructed. Some companies have so despaired of dealing with the cable chaos in existing buildings that moving the entire operation to another site and starting over can seem to be the least expensive alternative!

Cable chaos causes major financial and operational impacts. Inefficient management of communications assets can add up to many millions of dollars annually, and -- in the most serious scenario -- can bring business operations to a standstill. The consequences and costs include:

- * direct network maintenance and changed management costs
- * poor asset management
- * risk of business interruption.

Each move, add, or change in an undocumented network can cost between \$1000 and \$1500. Unfortunately, the communications manager can be certain that some portion of this expense is unnecessary. But without adequate information, there is no basis for finding and eliminating the waste. Expenses are sure to be high because project turnaround times are long and there is no management control.

From a wider facilities perspective, needless redundancy of cabling means wasted cableway space. Also, since there is probably a lack of information on the installed equipment base, utilization of all related communications assets will be inefficient. The problem can get so bad that ceilings literally fall from the weight of unused cabling.

SOLUTION: COMPREHENSIVE MANAGEMENT

The goal of a truly comprehensive communications management system (see Figure 1) is to link all communications resources -- not just cable -- to all corporate facilities for improved and complete asset management. The results can be reducing the cost of changes, shortening project turnover times, and providing management control for improved decision making.

Integration of telecommunications information within an organization-wide facilities management (FM) data base is needed because:

- * capital assets for telecommunications represent a significant portion of corporate investment in facilities
- * information on facilities, including floor plans, is crucial to communications planning and management
- * moves, adds, and changes in a network are often motivated by changes in facility utilization
- * an integrated (FM) data base allows efficient network administration across multiple sites.

A comprehensive communications management system can be achieved by capturing and representing all communications resources in a single, integrated data base. These comprehensive systems can be used to coordinate efficiently the processes of network planning, design, configuration, and change management. The move towards such integrated systems is now widespread. An example is Command (Communications Management and Design) from Isicad, which serves as a model for the following discussion.

INTEGRATION OF GRAPHICS AND TEXT

The comprehensive system's integrated data base contains information on communications resources as both graphic and alphanumeric (text) data.

A common area of understanding between telecommunications and FM functions is typically the floor plan (see Figure 2). Therefore, a CAD-based graphic system is a natural interface between facilities and telecommunications information. Network information shown graphically on floor plans includes devices and equipment, outlets, cable, and cableways.

RELATIONAL DBMS

In communications management systems, the advantages of a relational DBMS include linkage of graphics and text, data exchange, and flexible reporting. Integration of graphics and text saves work and enhances user understanding of the system and the network it represents. Changes to CAD drawings, such as floor plans, automatically update corresponding alphanumeric attributes. For example, adding the symbol of a computer terminal updates the corresponding text description and equipment listings. Conversely, changing an entry in an alphanumeric listing changes the affected symbols or cable routing on the drawings.

For compatibility among different computer environment and for portability of information, it is advantageous if the underlying relational DBMS in the system is a standard, commercially available package rather than a proprietary module. For ease of integration with existing FM and architectural CAD data bases, it is also important for the system to support graphic data exchange in different CAD data formats, such as IGES, SIF, and DXF. Thus, existing CAD drawings can be imported, and modified drawings can be exported. Benefits include savings in data entry time, as well as enhanced information sharing with other FM functions.

Relational DBMS technology has exceptionally flexible reporting capabilities. Attributes residing in the system can be queried, analyzed, and reported in a variety of ways. These reporting functions save time and labor by automating the paperwork requirements of routine communications management. The system can generate all kinds of documentation, including schematic diagrams, equipment and cable schedules, bills of materials, and work orders.

Reporting capabilities are further enhanced through built-in links to business graphics and spreadsheet modules (see Figure 3) -- an additional benefit of using an off-the-shelf data base. Use of business graphics and spreadsheet modules provides valuable tools for management decision making. For example, it is possible both to track historic resource utilization and to analyze that information as a basis for future planning. With these decision support tools, communications management can become truly proactive -- looking ahead of the usage curve -- rather than merely attempting to keep up with changes.

THE SYSTEM AT WORK

Specific communications management capabilities of comprehensive systems include:

- * creation and automated numbering of entities
- * data base query
- * patching
- * circuit reconfiguration
- * automatic work order creation
- * automatic schematic creation
- * reporting.

Creating and Automated Numbering of Entities

Working with the system begins with designing a network or documenting an existing one. Here, the extraordinary usefulness of the CAD capabilities becomes immediately obvious. Communication devices, cableways, cables, cable routes, and connections may be identified or defined, placed, and automatically numbered on CAD drawings. In the same operation, corresponding attributes are registered in the data base automatically.

CAD drawings can be either imported from other CAD systems, as described previously, or drawn within the communications management system when designing a network. To speed the process, equipment units are added to CAD drawings as symbols. The system's symbol library includes such items as specific PC models and telecom outlets. Symbols also may be defined by the user so that corporate standards can be followed. Each symbol has a set of attributes, including specifications such as manufacturer, model number, and cost which are added to the data base.

With these customized tools, the design phase moves quickly. Experience designing a network for 10-story bank facility has shown that a job typically requiring 24 man-years for the design phase could be reduced to 0.5 man-year with this CAD-based system.

Numbering entities is critical for tracking resources. Integrated systems assign numbers automatically to new entities and can adhere to the user's predefined format for compatibility with existing identification schemes. Again, the process of adding the symbols to the drawing also registers their identification numbers and attributes into the data base. Numbers then become important key field identifiers for subsequent searching within the DBMS.

As an example, consider the addition of two new PCs. The system prompts for a device number as a symbol for each unit that is placed in the drawing. If 20 terminals are already in the data base and two are being added, the system assigns the numbers 21 and 22. The corresponding telecom outlet symbols are added and numbered in similar fashion.

Keeping track of cable paths is fundamental to cable management decisions, and yet few systems until now have been able to perform this function. Through comprehensive systems, cable paths can be created and viewed graphically on the floor plan, as schematics or as listings of

routes and connections. To assist the designer in layout decisions, the system can flag design rule violations, such as when a cableway capacity is being exceeded as cable is routed. Further, the user can analyze the effects of disrupting a specific cableway, whether the disruption is intentional (as in making changes) or accidental (as in troubleshooting and disaster recovery).

In a single operation, cables can be laid out individually or, for example, as blocks of 50 twisted pairs. Either way, the system will assign and register an identifier for each cable. Cables may be of any type, including user-defined types, and may even be logical entities, such as microwave links.

Entities such as devices and cables can be added to drawing in any order. Establishing connections is then a straightforward procedure. For cable assignment, the system prompts with FROM and searches patch-panel entries in the data base for the first open socket. This selection is presented to the user, who can accept it or override it. The next prompt, TO, asks for the terminal device, which is selected simply by either pointing to the symbol with the graphics cursor or by entering the alphanumeric identifier. As selections are made, the system builds a cable routing schedule. This schedule contains total cable length, cable **type**, and **other** relevant data.

Data Base Query

As cable **layout** takes shape during the **network** design phase, the user is building an ongoing **data base**. This **data base** provides connectivity **information** that is essential both to completing the design phase and to managing subsequent changes (see Table 1).

Through the relational DBMS, the user can request information in a wide variety of ways from the data base. In the system used as a model, the nontechnical question-and-answer style of the DBMS query language is similar to IBM SQL, with which many users are already comfortable.

To further promote ease of use, the system's graphic capabilities make it possible for the user merely to point to objects on a drawing display to obtain attribute listings. The ability to view entities either graphically or as text increases understanding and helps assure that updates are kept current.

An example of interrogating the data base might be to obtain an equipment schedule, perhaps for asset evaluation. This schedule would show all devices and outlets within the system and their attributes, including cost. Also included can be an item for tracking online status, such as:

STATUS -- ORDERED, NOT INSTALLED

Other pertinent data on this listing might include the department, so that any moves, adds, and changes can be charged back to the appropriate cost center.

The user might also wish to determine whether a specific cableway is overloaded. For any cable in the network, the system can report its type, category, description, diameter, weight, and so on. Based on cable diameter, the system can calculate whether the cross-sectional area of a cableway is nearing capacity.

Data base query is especially valuable for trouble-shooting and disaster recovery. If a cable is damaged, the system can report instantly -- with flashing symbols on the graphic display -- all the affected resources, right down to specific sockets on patch panels and terminal outlets. FM resources that are critical to disaster recovery can also be tied to the data base, including fire, security, and environmental control systems.

Patching

Much of the effort in conventional cable management involves identifying and tracking patch-panel connections throughout the network. As shown in Figure 4, comprehensive systems can assign cables to specific ports on patch panels. Port assignments on each panel can be viewed and/or

changed at an administrator's terminal or at a graphic workstation. The system's implementation of patching is closely related to actual patching practice, making it easy and self-explanatory.

Circuit Reconfiguration

Moves, adds, and changes occupy much of the attention of communications managers. This work is costly, time-intensive, and error-prone. Through comprehensive communications management systems, proposed circuit changes can be made by disconnecting and reconnecting cables among devices on the drawings. Making changes on the drawing, in turn, automatically generates a schematic as well as a work order. To simplify the process and to prevent errors, menu-driven routines prompt the user through each step in responding to a request for a move, add, or change.

As moves, adds, and changes are entered into the data base, the system handles all aspects of updating and reconfiguring the network representation so that documentation always reflects the current status. A change log is maintained so that there is an orderly record of all transactions affecting the network.

Automatic Work Order Creation

Generation of work orders automatically by the system assures that reconfiguration requirements are communicated accurately to the employee who will actually do the physical work. Work orders can be tracked through their life cycles--from proposed, to approved, to implemented--thereby providing managers with an accurate picture of work in progress.

Work order document form an audit trail describing changes to the network over time. This documentation also serves as an aid to troubleshooting, a complete maintenance history, and a basis for network analysis.

Automatic Schematic Creation

A graphic schematic is an effective means of visualizing the effect of reconfigurations. The system can create graphic schematics "on the fly," or as reconfiguration functions are being performed.

Reporting

Data base query, as already discussed, can be a freeform, question-and-answer process. Some comprehensive systems also support more formal reporting, including both standardized, predefined reports and user-defined reports. The objective is to assist the user in making informed decisions, presenting findings to others, and documenting all aspects of communications management.

Standard reports can be selected from menus so that the infrequent user or manager can easily find items of interest. In select systems, even customized reporting capabilities exist.

DRAWING-LEVEL VS. PROJECT-LEVEL

Some currently available cable management systems amount to specialized enhancements of architectural CAD tools. There is not the capability to manage other types of resources, nor may there be any integration with the larger FM environment. Such systems might be termed drawing-level solutions, since the management perspective stops at the boundaries of an isolated floor plan.

Integration of floors, sites, and campuses can be achieved through the comprehensive systems described here, since they integrate views of both local and remote resources. For example, when disconnecting a terminal, the corresponding host and mux can be identified -- whether located on another floor, in another building, or on a different campus halfway across the world. Since the system tracks logical as well as physical links, microwave channels can also be traced. Tracing all affected resources permits their prompt reassignment, thereby promoting efficient utilization of assets. Such a comprehensive communications management systems can truly be called a project-level solution for dealing with cable chaos.

Gregory Bloom is the vice president of Isicad, Inc.

CAPTIONS: Comprehensive communications management system. (chart);
Implementation costs. (graph); Design cable schedule. (table)
COPYRIGHT 1990 Horizon House Publications Inc.

SPECIAL FEATURES: illustration; chart; graph; table

INDUSTRY CODES/NAMES: TELC Telecommunications

DESCRIPTORS: Relational data bases--Usage; Computer-aided design--Usage;
Telecommunication cables--Management

FILE SEGMENT: TI File 148

?

Set	Items	Description
S1	1140574	CONFIGURAT??? OR AUTOCONFIGURAT??? OR TOPOLOGY OR TOPOLOGI-ES OR LAYOUT? ?
S2	16859315	COMPUTER? ? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?
S3	8752399	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDENTIFI??????
S4	2616378	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BANK? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABANK? ? OR TABLE? ? OR FILE? ?
S5	4759186	SEARCH??? OR QUERY??? OR QUERI?? OR SCAN OR SCANNING OR SCANNED OR LOOKUP OR LOOK()UP OR RETRIEV??? OR LOCAT??? OR MATCH???
S6	25553876	GROUP? ? OR TYPE? ? OR RANGE? ? OR ARRAY OR COLLECTION OR CLASSIFIC????? OR DESCRIPTOR? ? OR CATEGOR??? OR SUBTYPE? ? OR MODEL? ? OR CLASS? ?
S7	18773495	SECOND??? OR OTHER OR ANOTHER OR ADDITIONAL OR EXTRA OR TWO OR 2ND
S8	4	BELMANAGE
S9	76042	(S5 AND (S1 OR S2) AND S3 AND S4 AND S7 AND S6) AND (PD<20-020407 OR PY<2003)
S10	625	S9 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?) (3N)S4)/TI
S11	84	S9 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?) (S4)/TI

? show files

File 2:INSPEC 1898-2006/Jan W3
(c) 2006 Institution of Electrical Engineers

File 6:NTIS 1964-2006/Jan W5
(c) 2006 NTIS, Intl Cpyrght All Rights Res

File 8:Ei Compendex(R) 1970-2006/Jan W5
(c) 2006 Elsevier Eng. Info. Inc.

File 34:SciSearch(R) Cited Ref Sci 1990-2006/Jan W5
(c) 2006 Inst for Sci Info

File 35:Dissertation Abs Online 1861-2006/Jan
(c) 2006 ProQuest Info&Learning

File 56:Computer and Information Systems Abstracts 1966-2006/Jan
(c) 2006 CSA.

File 57:Electronics & Communications Abstracts 1966-2006/Jan
(c) 2006 CSA.

File 60:ANTE: Abstracts in New Tech & Engineer 1966-2006/Jan
(c) 2006 CSA.

File 65:Inside Conferences 1993-2006/Jan W5
(c) 2006 BLDSC all rts. reserv.

File 94:JICST-EPlus 1985-2006/Nov W4
(c) 2006 Japan Science and Tech Corp(JST)

File 95:TEME-Technology & Management 1989-2006/Feb W1
(c) 2006 FIZ TECHNIK

File 99:Wilson Appl. Sci & Tech Abs 1983-2006/Jan
(c) 2006 The HW Wilson Co.

File 111:TGG Natl.Newspaper Index(SM) 1979-2006/Jan 30
(c) 2006 The Gale Group

File 144:Pascal 1973-2006/Jan W3
(c) 2006 INIST/CNRS

File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
(c) 1998 Inst for Sci Info

File 636:Gale Group Newsletter DB(TM) 1987-2006/Feb 03
(c) 2006 The Gale Group

?

BIBLIOGRAPHIC

NPL

Set	Items	Description
S1	275943	CONFIGURAT??? OR AUTOCONFIGURAT??? OR TOPOLOGY OR TOPOLOGI- ES OR LAYOUT? ?
S2	2860599	COMPUTER? ? OR PERIPHERAL? ? OR NETWORK? ? OR DP OR SYSTEMS OR TECHNOLOGY OR HARDWARE? ?
S3	2379331	INFORMATION OR INVENTORY OR INVENTORIES OR TRACK??? OR IDE- NTIFI??????
S4	719660	DATABASE? ? OR DATA() (BASE? ? OR UNIT? ? OR BASE? ? OR BAN- K? ? OR STRUCTURE? ? OR REPOSITORY? ?) OR DB? ? OR DATABANK? ? OR TABLE? ? OR FILE? ?
S5	1678081	SEARCH??? OR QUERY??? OR QUERI?? OR SCAN OR SCANNING OR SC- ANNED OR LOOKUP OR LOOK()UP OR RETRIEV??? OR LOCAT??? OR MATC- H???
S6	3935894	GROUP? ? OR TYPE? ? OR RANGE? ? OR ARRAY OR COLLECTION OR - CLASSIFIC????? OR DESCRIPTOR? ? OR CATEGOR??? OR SUBTYPE? ? OR MODEL? ? OR CLASS? ?
S7	7558603	SECOND??? OR OTHER OR ANOTHER OR ADDITIONAL OR EXTRA OR TWO OR 2ND
S8	564	(S5 AND (S1 OR S2) AND S3 AND S4 AND S7 AND S6 AND IC=(G06- F-007/00 OR G06F-017/30)) NOT (AD=20020407:20060206)
S9	33	((S5(100N) (S1 OR S2) (100N) S3(100N) S4(100N) (S7(3N) S6)) AND - IC=(G06F-007/00 OR G06F-017/30)) NOT (AD=20020407:20060206)
S10	13	(S8 AND ((S1 OR PERIPHERAL? ? OR NETWORK? ? OR SYSTEMS OR - TECHNOLOGY OR HARDWARE? ?) (3N) S4) /TI) NOT S9

? show files

File 347:JAPIO Nov 1976-2005/Oct(Updated 060203)

(c) 2006 JPO & JAPIO

File 350:Derwent WPIX 1963-2006/UD,UM &UP=200608

(c) 2006 Thomson Derwent

?

BIBLIOGRAPHIC
PATENT